Univerzitet u Beogradu
Matematički fakultet

Master rad

Principi programiranja korisničkog interfejsa na operativnom sistemu Android

Student:

Stefan Panić

Mentor:

dr Vladimir Filipović

Beograd, 2018.

Mentor: **prof. dr Vladimir Filipović**

Matematički fakultet, Univerzitet u Beogradu

Članovi komisije: **prof. dr Filip Marić**

Matematički fakultet, Univerzitet u Beogradu

mr Anđelka Zečević

Matematički fakultet, Univerzitet u Beogradu

***Datum odbrane*: _____**

Principi programiranja korisničkog interfejsa na operativnom sistemu Android

Apstrakt - Operativni sistem Android je ubedljivo najpopularniji operativni sistem za mobilne uređaje sa više od 2 milijarde aktivnih mesečnih korisnika i preko 3,3 miliona aplikacija na prodavnici aplikacija *Google Play*. Ovaj rad se bavi predstavljanjem osnovnih principa i radnih okvira koji se koriste pri dizajniranju i programiranju korisničkog interfejsa za Android. Takođe u radu je prikazana i implementacija aplikacije "*Učigraonica*" uz pomoć koje deca na zabavan i jednostavan način mogu savladati osnovne matematičke operacije i čitanje i pisanje na latinici ili ćirilici. Aplikacija je razvijana u cilju prezentovanja upotrebe velikog broja koncepta i radnih okvira navedenih u radu.

Ključne reči: Android operativni sistem, mobilni uređaji, korisnički interfejs, Android aplikacije, radni okviri, pametni telefon, razvoj

Principles of programming user interface on Android operating system

Apstract – Android operating system is by far the most popular operating system for mobile devices with more than 2 billion active monthly users and with more than 3,3 million apps on app store *Google Play*. This work focuses on presenting basic principles and frameworks used to design and develop user interface on Android. In addition, this work shows implementation of application "*Ucigraonica*" which helps children learn basic math operations, reading and writting in latin or cyrilic in a simple and entertaining way. This app is developed in order to demonstrate the use of many concepts and frameworks shown in this work.

Keywords: Android operating system, mobile devices, user interface, Android apps, frameworks, smartphone, development

Sadržaj

1 Uvod

Tržište „pametnih“ mobilnih telefona je u konstantom porastu. Prema podacima agencije „Counterpoint“ samo u toku 2017. godine prodato je oko 1,55 milijardi uređaja. Od toga 85,9% predstavlja udeo Android operativnog sistema, čime se pozicionira na prvo mesto rang liste po procentualnom udelu u tržištu mobilnih operativnih sistema. Takođe, dobar primer za rast upotrebe mobilnih uređaja je podatak da je u novembru 2016. godine više korisnika pristupalo Internetu preko mobilnih telefona i tableta nego preko klasičnih računara [7].

Jedan od razloga uspeha operativnog sistema Android je mogućnost pokretanja na uređajima različitih dimenzija, veličina i rezolucija ekrana i proizvođača. Aplikacije razvijane za ovaj operavini sistem su našle svoju primenu čak i na klasičnim laptop računarima, konvertibilnim i 3 u 1 računarima u okviru operativnog sistema Chrome OS. U martu 2018. godine broj aplikacija za Android OS je brojao preko 3,3 miliona. Sve ovo dovodi do povećane potražnje za programerima aplikacija za Android OS, dok dizajniranje i programiranje korisničkog interfejsa predstavlja veoma bitnu oblast pri razvoju mobilnih aplikacija. Uzimajući u obzir već pomenutu raznovrsnost uređaja zajedno sa heterogenošću korisnika „pametnih“ mobilnih telefona različitih uzrasta i zanimanja dovodi do pojave okvira, principa i preporuka za razvoj korisničkog interfejsa.

Napredak hardvera mobilnih uređaja dovodi i do stvaranja naprednih, komplikovanih i vizuelno updaljivih elemenata korisničkog interfejsa i samim tim omogućava razvojnim timovima stvaranje originalnih i jedinstvenih aplikacija. Samim tim povećava obim znanja i iskustva potrebnog za razvoj aplikacija.

Globalni trend rasta tržišta i potražnje za aplikacijama razvijanih za Android operativni sistem predstavlja motivaciju za pisanje ovog rada i za izučavanje ovih tehnologija, naručito programiranje i dizajniranje korisničkog interfejsa kao aspekta koji značajno može uticati na prihvaćenost i popularnost aplikacije.

U drugom poglavlju ovog rada biće ukratko opisana istorija operativnog sistema Android sa istorijatom verzija. Takođe će biti predstavljena arhitektura sistema kroz slojeve i radne okvire (*engl. Framework*).

U trećem poglavlju ovog rada biće razjašnjeno kako različitost uređaja utiče na razvoj aplikacija. Biće opisan skup pravila za dizajniranje korisničkog interfejsa poznat kao „Material Design“. Poglavlje će se baviti i osnovnim elementima aplikacije, životnim ciklusom i pogledima (*engl. Views*) kao glavnim alatima pri razvoju.

U četvrtom poglavlju biće prikazani obrasci, pravila, napredne tehnike, saveti i preporuke za razvoj aplikacija sa dobro dizajniranim i pre svega funkcionalnim korisničkim interfejsom.

U petom poglavlju će biti opisana implementacija aplikacije *„Učigraonica“*. Aplikacija je razvijana za decu uzrasta 6-12 godina koja omogućava deci da kroz igru nauče osnovne matematičke operacije, kao i da kroz pogađanje objekata sa slike uče čitanje i pisanje latinice i ćirilice. Kroz samu aplikaciju biće predstavljeni navedeni saveti, tehnike i koncepti iz poglavlja četiri. Aplikacija je otvorenog koda i sam izvorni kod se može naći na adresi: `https://github.com/panucci/Ucigraonica-Master`

U šestom poglavlju će biti izveden zaključak.

2 Operativni sistem Android

U ovom poglavlju će biti predstavljene osnovne informacije o operativnom sistemu Android, istorijat verzija, arhitektura sistema i biće navedene karakteristike i različitosti uređaja koje utiču na razvoj korisničkog interfejsa aplikacija.

2.1 Istorijat i verzije operativnog sistema

Android je operativni sistem otvorenog koda za mobilne uređaje razvijan od strane kompanije „Google“. Zasnovan je na izmenjenom Linux jezgru i drugom softveru otvorenog koda. Prvenstveno je dizajniran za rad na uređajima koji poseduju ekrane osetljive na dodir kao što su pametni telefoni i tableti, mada je u poslednjih par godina doživeo i ekspanziju na uređaje kao što su satovi, televizori, automobili, konzole za video igre i druge.

Suprotno verovanju da je Google tvorac, Android operativni sistem je nastao pod okriljem kompanije Android Inc, osnovane oktobra 2003. godine, koju je kompanija Google kupila za oko 50 miliona američkih dolara u julu 2005. godine. Inicijalna komercijalna verzija operativnog sistema je izdata u septembru 2008. godine kao Android 1.0. Od tada je operativni sistem prošao kroz niz značajnih ažuriranja i novih verzija, koji prati trend izdavanja nove verzije svakih godinu dana. Poslednja stabilna verzija nosi kodno ime Oreo (8.1.) i izdata je u decembru 2017. godine. Izvorni kod Android operativnog sistema je poznatiji kao AOSP (*engl. Android Open Source Project*) i on se razvija pod Apache licencom.

Android je ubedljivo najpopularniji i najkorišćeniji operativni sistem kada su mobilni uređaji u pitanju još od 2011. godine. Od maja 2017. ima više od 2 milijarde aktivnih mesečnih korisnika, dok na platformi Google Play postoji više od 3,3 miliona aplikacija. Sistem je podržan na više od 100 svetskih jezika. [6]

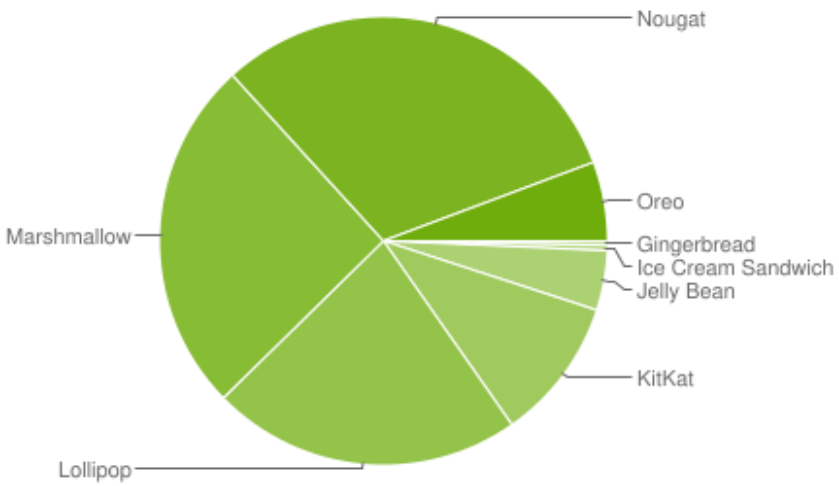
Raznovrsnost uređaja i brza adaptacija su doveli do brzog porasta broja korisnika, što je izazvalo potencijalne probleme za programere. Aplikacije moraju da podrže različite veličine ekrana, rezolucije, hardverske ili softverske tastature, senzore itd. To značajno otežava testiranje aplikacija u svim navedenim uslovima. Apstrakcijom hardverskih

različitosti Android OS pokušava da programiranje aplikacija učini nezavisnim, ili u što manjoj meri zavisnim, od samih karakteristika uređaja. Takođe izlazak novih verzija OS-a omogućava kompatibilnost aplikacija razvijanih za ranije verzije korišćenjem opštih API-a (*engl. Application Programming Interface*) koje Android nudi [3].

U tabeli 2.1. i na slici 2.1. je dat prikaz udela verzija operativnog sistema na tržištu, dok je u tabeli 2.2. dat kratak pregled verzija.

Verzija	Kodno ime	Udeo
2.3.3. - 2.3.7.	Gingerbread	0,3%
4.0.4. - 4.0.4.	Ice Cream Sandwich	0,4%
4.1.x 4.2.x 4.3	Jelly Bean	1,5% 2,2% 0,6%
4.4	KitKat	10,3%
5.0 5.1	Lollipop	4,8% 17,6%
6.0	Marshmallow	25,5%
7.0 7.1	Nougat	22,9% 8,2%
8.0 8.1	Oreo	4,9% 0,8%

Tabela 2.1. Udeo verzija operativnog sistema na tržištu[8]



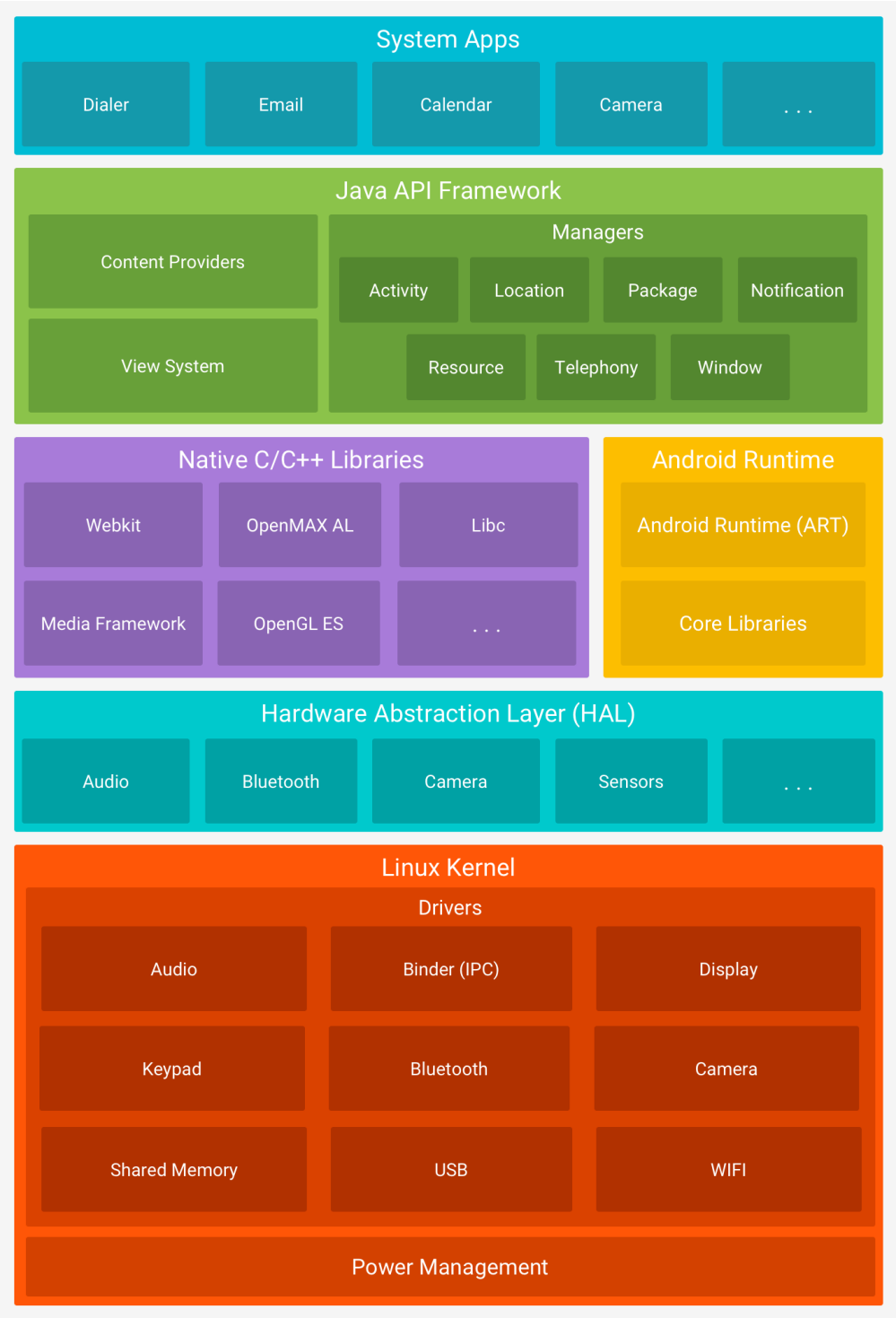
Slika 2.1. Udeo verzija operativnog sistema na tržištu[8]

Verzija	Kodno ime	Pregled izmena i datum
1.0	/	September 2008. - *Android Market*, *Web browser*, podrška za kameru, Google servisi (Gmail, Maps, Contacts, Sync…), media plejer, sistem obaveštenja (*engl. notifications*).
1.1	/	Februar 2009. - Unapređene SMS funkcionalnosti, SDK 1.1.
1.5	Cupcake	April 2009. - SDK 1.5. Snimanje i reprodukcija MPEG-4 i 3GP video formata, widgets, animacije.
1.6	Donut	Septembar 2009. - Proširen skup podržanih komandi pokretom, pretraga glasom, podrška za uređaje sa WVGA rezolucijom ekrana.
2.0	Eclair	Oktobar 2009. - Nove opcije kamere, Bluetooth 2.1, unapređenje softverske tastature, rečnici, SDK 2.1.
2.2	Froyo	Januar 2010. - Poboljšanje API-a, Chrome V8 JS, podrška za *Adobe Flash*, WiFi hotspot.
2.3	Gingerbread	Decembar 2010. - Upravljanje radnom memorijom, žiroskop, *NFC*, nadogradnja korisničkog interfejsa.
3.0	Honeycomb	Februar 2011. - Uglavnom unapređenje podrške za tablete, multitasking, podrška za procesore sa više jezgara.
4.0	Ice Cream Sandwich	Oktobar 2011. - Softverski navigacioni tasteri na pametnim telefonima, prečice do aplikacija dok je uređaj zaključan, otključavanje licem, statistike potrošnje podataka mobilnog Interneta, editor fotografija.
4.1	Jelly Bean	Jun 2012. - Optimizacija grafičkog interfejsa, proširivanje obaveštenja, unapređenje aplikacije za kameru, kreiranje više korisnika na jednom uređaju, kontrole iz sekcije statusne linije (*Quick Settings*), *BLE*.
4.4	KitKat	Septembar 2013. - Optimizacija za uređaje sa manje radne memorije, multitasking taster menja meni taster, novi radni okvir za grafički interfejs, novi *ART* (*Android Runtime*).
5.0	Lollipop	Jun 2014. - ART sa novim principom kompilacije, podrška za 64-bitnu arhitekturu, *Material Design*, projekat *Volta* (optimizacija potrošnje baterije), oficijalna podrška za više SIM kartica.
6.0	Marshmallow	Maj 2015. - *Doze* (dodatna optimizacija potrošnje baterije), podrška za čitač otiska prsta, USB-C, vertikalna navigacija aplikacija, 4K rezolucija.
7.0	Nougat	Avgust 2016. - Kalibracija boja, zumiranje u aplikacijama, podrška za prikaz više aplikacija u isto vreme (*split screen*), novi JIT kompajler, bolji menadžer datoteka.
8.0	Oreo	Avgust 2017. - Projekat *Table,* modularna arhitektura, redizajn delova korisničkog interfejsa, unapređenje sistema obaveštenja, *Google Play Protect*, restruktuiranje podešavanja, manji *boot time*.

Tabela 2.2. Istorijat verzija Android operativnog sistema.

2.2 Arhitektura operativnog sistema

Android je softver otvorenog koda zasnovan na Linux jezgru, napravljen za rad na velikom spektru uređaja različitih karakeristika i primena. Na slici 2.2. prikazane su osnovne komponente arhitekture Android platforme.



Slika 2.2. Arhitektura operativnog sistema Android

Osnovni slojevi arhitekture su: Linux jezgro (*engl. Linux kernel*), Sloj apstrakcije hardvera (*engl. Hardware abstraction layer – HAL*), C/C++ biblioteke, Android izvršno okruženje (*engl. Android runtime*), Java API radni okvir (*engl. Java API framework*) i Sistemske aplikacije (*engl. System Apps*).

Linux jezgro predstavlja najniži sloj i temelj Android platforme. Ovaj sloj omogućava osnovne funkcionalnosti kao što su rad sa nitima procesora, upravljanje unutrašnjom memorijom kao i ključne sigurnosne opcije. Takođe, pruža mogućnost proizvođačima da razvijaju drajvere (*engl. Driver*) za svoje uređaje na već poznatom Linux jezgu.

HAL pruža komunikaciju između hardvera i viših slojeva arhitekture. Ovaj sloj se sastoji od više modula (biblioteka), dok svaki modul implementira interfejs za rad sa određenom hardverskom komponentom (npr. zvučnikom, NFC modulom, *Bluetooth* ili *WiFi* modulom, žiroskopom itd.).

Android Runtime ART je kreiran sa mogućnošću pokretanja više virtualnih mašina na uređajima sa malo radne memorije. Od API-a 21 (Android 5.0 Lollipop) svaka aplikacija pokreće svoj proces i ima svoju instancu ART-a, dok je pre API-a 21 korišćen Dalvik. Osnovne osobine Android Runtime-a: kompilacija AOT (*engl. Ahead of time*) i JIT (*engl. Just in time*), optimizovani GC (*engl. Garbage collector*), detaljna dijagnostika, sistem izuzetaka i bolja podrška pri otklanjanju grešaka (*engl. Debugging*).

Nativne C/C++ biblioteke predstavljaju skup biblioteka koje su napisane programskim jezicima C i C++. Drugi slojevi arhitekture su napisani oslanjajući se na ovaj sloj i na nativne biblioteke ova dva jezika. Android omogućava aplikacijama pristup ovim bibliotekama preko Java API radnog okvira.

Java API radni okvir sadrži skup funkcionalnosti Android operativnog sistema dostupnog kroz API napisan u jeziku Java. Ovaj sloj omogućava ponovnu upotrebu koda servisa, komponenti i modula koji sadrže:

- Sistem pogleda (*engl. View system*) koji se koriste za dizajniranje korisničkog interfejsa (liste, dugmad, polja sa tekstom, polja za unos itd.).
- Menadžer Android resursa (*engl. Resource manager*) omogućava upotrebu datoteka sa niskama, fontovima, konstantama, XML-ovima itd.

- Menadžer obaveštenja (*engl. Notification manager*) koji omogućava aplikacijama da prikažu korisniku razna obaveštenja i upozorenja.
- Snadbevač sadržaja (*engl. Content Provider*) koji pruža pristup podacima iz drugih aplikacija kao što su mape, kontakti, pozivi itd.

Sistemske aplikacije predstavljaju skup bazičnih aplikacija (SMS, kalendar, internet pretraživač, softverska tastatura, kontakti itd.). Kod operativnog sistema Android korisnik nije u obavezi da koristi ugrađene sistemske aplikacije, već može koristiti aplikacije koje je sam preuzeo ili instalirao. Tako da sistemske aplikacije imaju isti status kao i aplikacije naknadno instalirane. [9]

2.3 Specifikacije i raznovrsnost uređaja

Broj uređaja sa operativnim sistemom Android je u konstantnom porastu. Prema podacima kompanije *Google* trenutno je aktivno preko 1,4 milijardi uređaja, većinom mobilnih telefona i tablet računara. Jedan od glavnih razloga za uspeh Androida je raznovrsnost uređaja različitih proizvođača.

Neki od proizođača uređaja koji pokreću operativni sistem Android su: Samsung, LG, HTC, Xiaomi, Huawei i mnogi drugi. Unapređenje i optimizacija hardvera, rasprostranjenost tržišta, potražnja za uređajima različitih osobina diktiraju smer razvoja operativnog sistema, tako da on podržava veliki spektar hardvera prenosnih uređaja. Iz tog razloga Android ima široku oblast primene. Tako da danas postoje uređaji sa laserskim barkod skenerima za magacinsko poslovanje, fitnes narukvice, "pametni" satovi, laptop računari, TV Box uređaji koji podržavaju ogroman broj Android aplikacija, televizori, konzole za video igre, mobilni telefoni, tableti i mnogi drugi uređaji. Obrazovanje, sport, zabava, media plejeri, komercijala, analiza poslovanja su samo neke od oblasti gde je Android našao primenu.

U poslednjih 5 godina došlo je i do naglog napretka hardvera koji proizvođači koriste u svojim uređajima. Nije retkost da "pametni" mobilni telefoni dolaze sa osmo-jezgarnim procesorom, 4,6 ili 8 gigabajta radne memorije. Za potrebe ovog rada, kao najvažnije karakteristike, izdvajaju se dimenzije i rezolucija ekrana, kao i način upotrebe uređaja, pre svega da li će uređaj biti korišćen u vertikalnom ili horizontalnom režimu.

Dimenzije ekrana uređaja su najčešće izražene u inčima, i predstavljaju dužinu dijagonale samog ekrana. Kod Android mobilnih telefona trenutni trend je da dijagonala ekrana iznosi između 4,5 inča do 6 inča, dok se kod tablet računara taj broj kreće od 7 pa sve do 13 inča. Rezolucija ekrana predstavlja broj nezavisnih piksela (*engl. Pixel, skraćeno px*), odnosno tački, po horizontalnoj i vertikalnoj osi. Izražava se kao M x N, gde M predstavlja broj piksela po horizontalnoj, a N broj piksela po vertikalnoj osi. Kod Android uređaja najčešće rezolucije su 720p (1280 x 720), 1080p (1920 x 1080), 2K (2560 x 1440) ali i sve češće rezolucije koje imaju odnos piksela 18:9 ili 18,5:9 kao što je npr. 2160 x 1080.

Sve ovo dovodi do značajnog otežavanja razvoja aplikacija, kao i samog testiranja, odnosno nemogućnosti testiranja na svim mogućim uređajima. Pozicioniranje i veličina elemenata grafičkog interfejsa su u velikoj korelaciji sa veličinom dijagonale ekrana pa zahteva dodatno planiranje pri razvoju. Da bi se olakšalo dizaniranje, planiranje i programiranje korisničkog interfejsa uvode se mere *gustine piksela* (*engl. Pixel density*) i *broj piksela po inču* DPI (*engl. Dots per inch*) kao i dve jedinice mere:

- DIP ili DP (*engl. Density independent pixel*) predstavlja apstraktnu jedinicu (virtuelni piksel) koja je izvedena iz gustine piksela ekrana. Za 1 DP je uzeta veličina jednog piksela na ekranu sa gustinom piksela 160 DPI. Odnos piksela i DIP će se menjati sa porastom gustine ekrana, ali ne obavezno u direktnoj proporciji. Preporuka je koristiti ovu jedinicu pri definisanju elemenata grafičkog korisničkog interfejsa iz razloga što je nezavisna od gustine piksela ekrana.
- SIP ili SP (engl. Scale independent pixel) je takođe apstraktna jedinica slična DP ali je dodatno skalirana izborom veličine fonta od strane korisnika. Preporuka je korisiti ovu jedinicu pri definisanju veličine fonta tekstualnih polja tako da utiču i gustina piksela ekrana i korisnički izbor [4].

Pored navedenih jedinica, programerima je na raspolaganju kreiranje različitih direktorijuma u kojima će biti definisane XML datoteke za ekrane različitih gustina piksela, kao i različite orijentacije ekrana, što će detaljnije biti opisano u poglavlju 3.

U tabeli 2.3. dat je prikaz prosečnih dimenzija i rezolucija ekrana u poslednjih deset godina [5].

Godina	Prosečna rezolucija (predstavljeno u odnosu 16:9)	Prosečna veličina dijagonale (u inčima)	Prosečna gustina piksela
2007	387 x 218	2.3	171
2008	433 x 243	2.5	182
2009	539 x 303	2.8	197
2010	623 x 350	3.1	204
2011	737 x 414	3.6	216
2012	897 x 504	4.1	226
2013	1314 x 739	4.6	294
2014	1457 x 819	5	309
2015	1808 x 1017	5.2	375
2016	1753 x 986	5.3	368
2017	2011 x 1131	5.4	411

Tabela 2.3. Prosečne dimenzije i karakteristike ekrana Android mobilnih telefona.

3 Osnovni principi programiranja korisničkog interfejsa

U ovom poglavlju će biti predstavljen skup pravila i preporuka poznatiji kao *Material Design,* zatim će detaljnije biti objašnjeni osnovni gradivni elementi aplikacija kao i njihovi životni ciklusi. Takođe će biti prikazan način na koji se definiše izgled i raspored elemenata korisničkog interfejsa zajedno sa osnovnim pogledima (*engl. Views*).

3.1 Razvojno okruženje, programski jezici

Programiranje Android aplikacija je prvenstveno zamišljeno i ostvareno upotrebom jezika Java i *Android Studio* razvojnog okruženja, koji i danas predstavljaju najčešće korišćene alate za razvoj. Potreba da se razvijaju aplikacije za više operativnih sistema mobilnih uređaja (iOS, Windows) dovodi do pojave *Xamarin* platforme koja nudi rešenje ovog problema. *Xamarin* je razvijan pod licencom otvorenog koda pod vlasništvom firme Microsoft. Jezik koji se koristi je C# i omogućava potpuno istu funkcionalnost kao i Java uz prepoznatljive razlike u samoj sintaksi, što pruža mogućnost programerima koji već poznaju C# da razvijaju aplikacije za Android OS. *Xamarin* se koristi kao dodatak alatu *Visual Studio*, iako je do skoro postojao i alat *Xamarin Studio* ali je njegov razvoj prekinut. Svi navedeni principi, preporuke i saveti se mogu primeniti nezavisno od jezika koji se koristi za razvoj. Primeri koda će biti dati u jeziku C#.

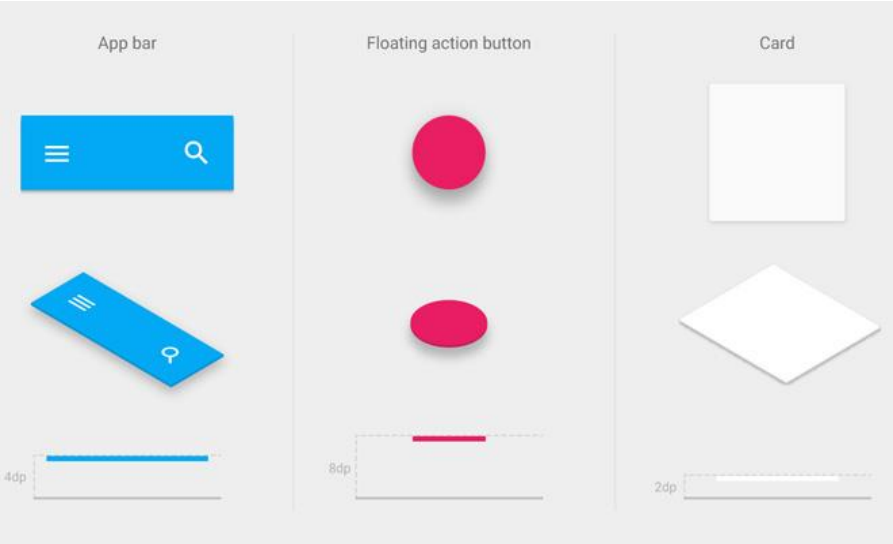
3.2 Material Design

Material Design je dizajnerski jezik osmišljen u kompaniji Google i predstavljen 2014. godine. Iako je prvenstveno planiran za operativni sistem Android i uređaje sa ekranima osetljivim na dodir, Material Design je našao primenu i u veb aplikacijama. Svoje prvo pojavljivanje je imao u Android verziji 5.0 (kodno ime Lollipop) kroz API biblioteke verzije 21.

Material Design je veoma opširan i detaljan skup pravila koji za cilj ima da postavi pozadinski sistem (temelj) koji omogućava razvoj korisničkog interfejsa za razne platforme i uređaje. Ujedno daje programerima fleksibilnost za inovacije i kreiranje spoja klasičnih principa dobrog korisničkog interfejsa sa mogućnostima tehnologije.

3.2.1 Osobine materijala

Jezik je zasnovan i inspirisan materijalima, strukturama i teksturama iz realnog sveta, načinom na koji reflektuju svetlost ili stvaraju senke. Ovi materijali su u digitalnom svetu predstavljeni kao homogeni, čvrsti objekti sačinjeni od piksela. Material Design je zasnovan na slojevima koji mogu biti predstavljeni kao listovi papira u realnom svetu. Slojevi (materijali) ne mogu prolaziti jedan kroz drugi. Oni su odvojeni upotrebom senki i percepcijom dubine. Time se postiže da korisnik zna koji element pripada kom sloju, sa kojim slojem je moguće interagovati, postiže se hijerarhija slojeva preklapanjem i prekrivanjem kao i u fizičkom svetu. Digitalni materijalni objekti se opisuju sa tri veličine, kao i u trodimenzionalnom prostoru, X, Y i Z osom, odnosno dužinom i širinom dok treća dimenzija (*engl. Elevation*) predstavlja koliko je taj objekat izdignut u odnosu na objekat (sloj) ispod njega. Debljina svakog materijala je 1dp. Objekti mogu menjati svoje dimenzije i oblik zadržavajući ostale karakteristike materijala. Svi tekstualni podaci, slike, video ili bilo koji drugi sadržaj leži u ravni sa slojem na koji se postavlja, kao što se u realnom svetu tekst štampa na papir. Sadržaj i sloj materijala na koji je postavljen predstavljaju celinu, i kao celina reaguju na interakciju korisnika.

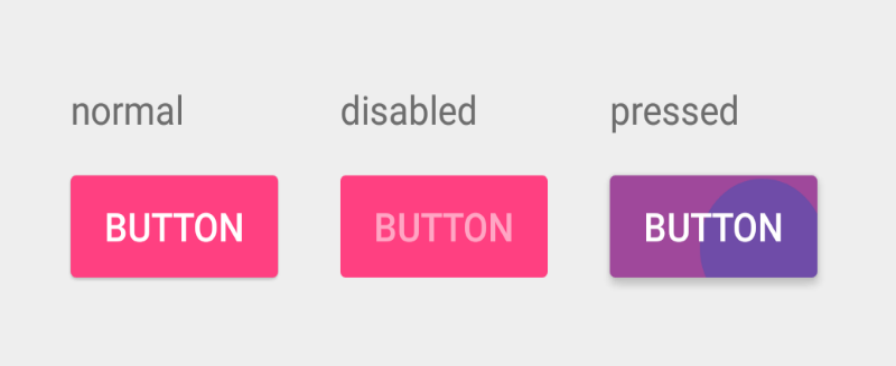


Slika 3.1. Prikaz nekoliko materijalnih objekata.

3.2.2 Animacije, ponašanje, interakcija, navigacija

Korisnici mogu imati interakciju sa elementima kroz dodir, prevlačenje ili druge pokrete. Animacije i pokreti predstavljaju važnu stavku ovog dizajnerskog jezika jer

daju kontekst, smisao i pružaju odgovor interfejsa korisniku. Korisničke akcije ne mogu prolaziti kroz slojeve materijala, tj. ukoliko korisnik dodirne određeni deo ekrana, na to može reagovati samo sloj koji se nalazi na vrhu. Svaka korisnička akcija mora imati odgovarajuću reakciju interfejsa. Reakcije interfejsa se najčešće postižu promenom boje ili oblika, senki odnosno izdignuća objekta u odnosu na sloj iza, premeštanjem položaja objekta na ekranu, zatamnjivanjem pozadinskog sloja, jednostavnom i kratkom animacijom na dodir objekta itd. Animacije i promene u interfejsu treba da odgovore na pitanja korisnika kako i odakle je došao do trenutnog stanja aplikacije, kao i da korisniku nedvosmisleno odgovore da je njegova akcija registrovana. Primer jedne takve animacije se može videti na slici 3.2. Navigacija i kretanje kroz aplikaciju mora biti jednostavno i intuitivno.

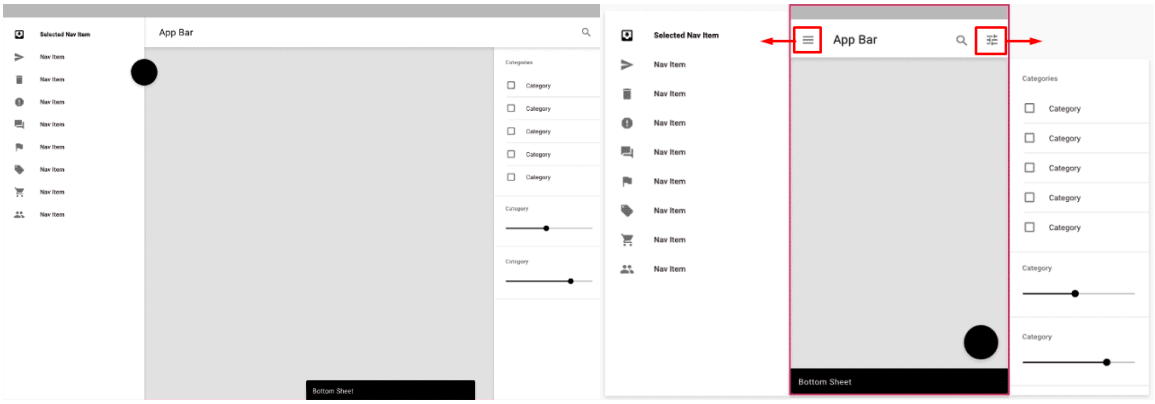


Slika 3.2. Postizanje reakcije interfejsa pri dodiru korisnika na dugme.

3.2.3 Raspored elemenata interfejsa

Osnovni principi pri organizaciji rasporeda objekata i elemenata korisničkog interfejsa su konzistentnost, predvidivost i odzivnost. Za postizanje konzistentnosti koristi se mera koja je nezavisna od gustine piksela (dp), tako da elementi proporcijalno zauzimaju isti prostor u odnosu na veličinu i rezoluciju ekrana. Predvidivost se postiže upotrebom jasno definisanih regija i celina. Prilagodljivost predstavlja mogućnost interfejsa da se adaptira u odnosu na nastale promene. Okretanjem ekrana u horizontalni ili vertikalni položaj ili upotrebom aplikacije na uređaju sa znatno većim ekranom treba ispratiti promenom broja i rasporeda elemenata na ekranu, rastojanjem između elemenata, promenom margina, oblika, boje ili pozicije elementa. Kako bi se korisniku olakšala upotreba aplikacije veoma je bitna veličina elemenata sa kojima on može da ima interakciju. Preporučena minimalna dimenzija je 48dp x 48dp. Upotreba

regija sa navigacionim elementima, sadržajem ili linijama menija bi trebalo da bude konzistentna i uniformna u toku rada sa aplikacijom na jednom uređaju, dok bi se vidljivost i pristupačnost regija razlikovale na uređajima sa različitim ekranima. Odnosno na uređajima sa manjim ekranima bi pristupačnost regijama za navigaciju i akcije menija bila dostupna kroz dugmad ili bočne menije. Primer se može videti na slici 3.3.



Slika 3.3. Implementacija regija na uređajima sa različitim ekranima.

3.2.4 Sistem boja

Material Design takođe sadrži savete vezano za izbor boja i upotrebu boja u aplikaciji. Glavno pravilo je izbor primarne i sekundarne boje koje bi trebalo da oslikavaju brend, ukoliko on postoji, ili na bilo koji drugi način zavise od ciljane grupe korisnika i funkcije aplikacije. Tako će aplikacije za decu koristiti više življih boja, dok će poslovne aplikacije koristiti suptilnije boje. Upotrebom boja se može vršiti označavanje neaktivnih elemenata, relacije elemenata, naglašavanje bitnih elemenata, objekata ili upozorenja. Izbor boja ne sme da ugrozi vidljivost teksta, ikonica ili bilo kog drugog sadržaja. Primarne i sekundarne boje se koriste zajedno sa njihovim svetlijim i tamnijim varijantama.

Primarna boja predstavlja dominantnu boju aplikacije i najčešće je korišćena boja u svim segmentima. Tamnije i svetlije varijante primarne boje se mogu koristiti da odvoje različite elemente u okviru jedne kontrole, odvajanje statusne linije od aplikacione linije sa alatima (*engl. App Bar*) kao i za naglašavanje linija sa opcijama menija. Moguć je izbor dve primarne boje, najčešće iz slične palete.

Sekundarna boja je opciona i može imati svetlije i tamnije varijante. Ona pruža više mogućnosti za kreiranje unikatnih i prepoznatljivih aplikacija. Osnovne upotrebe su za: plutajuću dugmad (*engl. Floating Action Button – FAB*) o kojima će biti reč u narednim poglavljima, obeležavanje označenog teksta, bojenje naslova, elemenata koje treba naglasiti, bojenje klizača (*engl. Sliders*), prekidače (*engl. Switches*).

Boje koje se koriste za prikaz greški, upozorenja, boju teksta obično nisu vezane za izbor primarne i sekundarne boje. Za greške se koristi nijansa crvene boje (#B00020), za boju teksta delimično prozirna bela ili crna boja u zavisnosti od boje pozadine. Prozirnost boje se određuje u zavinosti od aktivnosti kontrole (od 60% za neaktivne do 100% za aktivne elemente).

Light Blue 50	#E1F5FE
100	#B3E5FC
200	#81D4FA
300	#4FC3F7
400	#29B6F6
500	#03A9F4
600	#039BE5
700	#0288D1
800	#0277BD
900	#01579B

Cyan 50	#E0F7FA
100	#B2EBF2
200	#80DEEA
300	#4DD0E1
400	#26C6DA
500	#00BCD4
600	#00ACC1
700	#0097A7
800	#00838F
900	#006064

Teal 50	#E0F2F1
100	#B2DFDB
200	#80CBC4
300	#4DB6AC
400	#26A69A
500	#009688
600	#00897B
700	#00796B
800	#00695C
900	#004D40

Red 50	#FFEBEE
100	#FFCDD2
200	#EF9A9A
300	#E57373
400	#EF5350
500	#F44336
600	#E53935
700	#D32F2F
800	#C62828
900	#B71C1C

Pink 50	#FCE4EC
100	#F8BBD0
200	#F48FB1
300	#F06292
400	#EC407A
500	#E91E63
600	#D81B60
700	#C2185B
800	#AD1457
900	#880E4F

Purple 50	#F3E5F5
100	#E1BEE7
200	#CE93D8
300	#BA68C8
400	#AB47BC
500	#9C27B0
600	#8E24AA
700	#7B1FA2
800	#6A1B9A
900	#4A148C

Slika 3.4. Prikaz nekoliko paleta boja sa tamnijim i svetlijim varijantama, kao i bojom teksta koja im odgovara

U prethodnom delu teksta su predstavljeni osnovni principi, pravila i načela *Material Design* jezika. Sam jezik je veoma detaljno i temeljno objašnjen sa oblastima koje nisu pokrivene u ovom radu kao što su tipografija, pokreti, izbor ikonica, formati podataka, upotreba slika ali i mnoge druge.

Za opširnije čitanje, kao glavni izvor, posetiti veb stranicu *Material Design*. [11]

3.3 Osnovni gradivni elementi

3.3.1 Aktivnosti

Osnovni gradivni element korisničkog interfejsa je Aktivnost (*engl. Activity*). Aktivnost se može posmatrati kao prozor ili dijalog kod standardnih desktop aplikacija ili kao

jednu veb stranu u veb aplikacijama. Svrha Aktivnosti je da omogući interakciju sa korisnikom tako što kreira prozor u koji se smeštaju elementi korisničkog interfejsa. Najčešće se prikazuju kao prozor koji zauzima čitav ekran, mada mogu biti i u obliku dijaloga ili kao plutajući prozor (*engl. Floating Window*). U aplikaciji može postojati samo jedna Aktivnost, mada je praksa da aplikacija koristi više Aktivnosti. Svaka aplikacija mora da definiše korenu Aktivnost, odnosno Aktivnost koja će se prva prikazati pri pokretanju aplikacije. Funkcionalnost Aktivnosti je dostupna kroz nasleđivanje klase *Android.App.Activity* (ili *AppCompatActivity* u novijim verzijama biblioteka). Primer definisanja jedne Aktivnosti je dat na slici 3.5.

```
[Activity(Label = "ExampleActivity")]
public class ExampleActivity : Activity
{
    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);

        // dodeljivanje izgleda Aktivnosti
        SetContentView(Resource.Layout.example_activity_layout);
    }

    protected override void OnResume()
    {
        base.OnResume();
        Toast.MakeText(this, "OnResume", ToastLength.Short).Show();
    }

    protected override void OnPause()
    {
        base.OnPause();
        Toast.MakeText(this, "OnPause", ToastLength.Short).Show();
    }
}
```

Slika 3.5. Primer definisanja jedne Aktivnosti sa prevazilaženjem metoda OnResume i OnPause

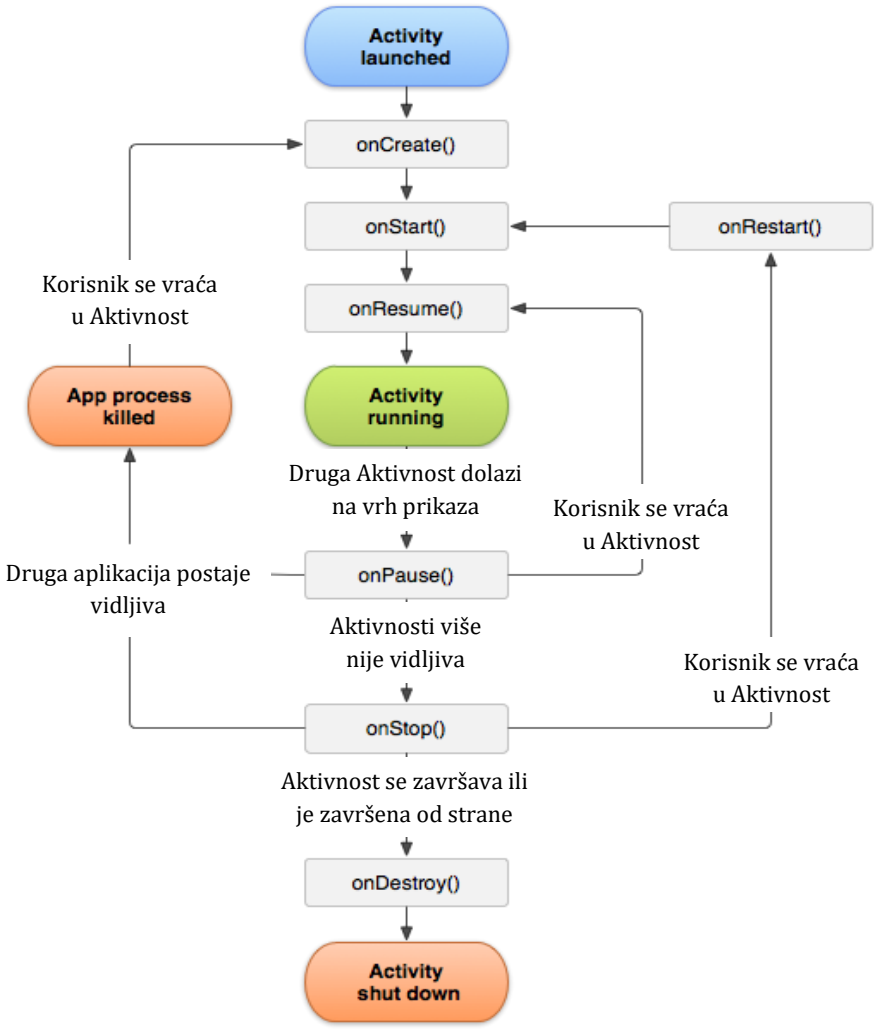
Aktivnost ima svoj životni ciklus (*engl. Lifecycle*). Veoma je bitno dobro razumeti životni ciklus Aktivnosti, jer nasleđivanjem klase *Android.App.Activity* i prevazilaženjem (*engl. Override*) metoda životnog ciklusa postiže željeno ponašanje i izgled Aktivnosti.

Delovi životnog ciklusa jedne Aktivnosti:

- Ceo životni vek Aktivnosti se odvija između poziva metoda `OnCreate()` i `OnDestroy()`. U metodi `OnCreate` će biti inicijalizovano stanje Aktivnosti sa potrebnim resursima, koji će biti pušteni pozivom `OnDestroy()`.

- Aktivnost je vidljiva između poziva metode `OnStart()` i odgovarajućeg poziva `OnStop()`. U tom periodu korisnik može da vidi Aktivnost na ekranu uređaja iako se može dogoditi da Aktivnost nije na vrhu prikaza, odnosno može biti zaklonjena dijalogom ili plutajućim prozorom. Između ova dva metoda Aktivnost čuva stanje, promenljive i resurse koji su joj potrebni.
- Između poziva `OnResume()` i `OnPause()` Aktivnost se nalazi u prvom planu, odnosno nalazi se na vrhu prikaza na ekranu. Tokom ovog perioda Aktivnost često može preći u stanje pauze, na primer kada uređaj "zaspi". Preporuka je da se u ovim metodama ne nalazi puno zahtevnog koda iz razloga što se često pozivaju.

Životni ciklus jedne Aktivnosti je dat na slici 3.6.



Slika 3.6. Prikaz životnog ciklusa jedne Aktivnosti

Prevazilaženjem ovih metoda može se upravljati ponašanjem Aktivnosti. Takođe omogućava programerima reagovanje na promene u aplikaciji ili na samom uređaju (okretanje ekrana, zaključavanje itd.), olakšava kontrolu promenljivih i resursa u zavisnosti da li je pozvan dijalog ili se Aktivnost završava.

U Android operativnom sistemu prilikom okretanja ekrana, iznenadnog prikaza druge aplikacije (npr. prilikom telefonskog poziva) ili ukoliko sistem želi da oslobodi radnu memoriju dolazi do gubitka trenutnog stanja promenljivih i resursa, tako da programer u tim situacijama mora voditi računa šta želi da sačuva. Programeru su na raspolaganju metode *OnSaveInstanceState(Bundle saveInstaceState)* i *OnRestoreInstanceState(Bundle savedInstaceState)*, koje se pozivaju kada dođe do promene stanja Aktivnosti. *Bundle* predstavlja objekat sličan kolekciji u koji je moguće smestiti i iz njega pročitati promenljive. Praksa je da se čitanje sačuvanih promenljivih vrši u pozivu *OnCreate(Bundle savedInstanceState)* umesto u *OnRestoreInstanceState(Bundle savedInstaceState)*. Primer upotrebe navedenih metoda dat je na slici 3.7.

Čest je slučaj da u toku korišćenja aplikacije korisnik prelazi iz jedne Aktivnosti u drugu, na primer pritiskom na dugme "nazad" (*engl. Back*) ili jednostavnom navigacijom kroz aplikaciju. Jedna Aktivnost može pokrenuti drugu korišćenjem poziva metoda *StartActivity(Intent intent)* ili *StartActivityForResult(Intent intent, int result_code)*. *Intent*, odnosno namera, sadrži informacije o Aktivnosti koju treba pokrenuti i opciono podatke koje treba proslediti novoj Aktivnosti. Metoda *StartActivityForResult* omogućava prenos podataka između pozvane Aktivnosti i Aktivnosti koja je izvršila poziv kada se završi životni vek pozvane Aktivnosti. Primer poziva Aktivnosti pomoću metode *StartActivity* i *StartActivityForResult* dat je na slici 3.8.

Aktivnosti se prilikom pokretanja nove Aktivnosti ređaju na stek. Stara Aktivnost gubi fokus, zaustavlja se i stavlja na vrh steka, dok nova Aktivnost postaje vidljiva korisniku. Ukoliko korisnik pritisne dugme za nazad ili ukoliko dođe do prekida Aktivnosti, skida se prva Aktivnost sa steka i ona postaje trenutno aktivna. Ukoliko se stek isprazni dolazi do prekida izvršavanje aplikacije. Na steku su moguće samo operacije dodavanja na vrh, i skidanja poslednje Aktivnosti sa vrha steka, nije moguće vršiti reogranizaciju i mešanje steka.

```
[Activity(Label = "ExampleActivity")]
public class ExampleActivity : Activity
{
    int brojac = 11;
    string ime_korisnika = "Marko Markovic";

    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);

        SetContentView(Resource.Layout.example_activity_layout);

        // vraćanje podataka se može izvršiti u OnCreate ili u
        // OnRestoreInstanceState
        if (savedInstanceState != null)
        {
            brojac = savedInstanceState.GetInt("brojac", 0);
            ime_korisnika = savedInstanceState.GetString("ime_korisnika", "");
        }
    }

    // pamćenje bitnih podataka u okviru objekta Bundle
    protected override void OnSaveInstanceState(Bundle outState)
    {
        base.OnSaveInstanceState(outState);
        outState.PutInt("brojac", brojac);
        outState.PutString("ime_korisnika", ime_korisnika);
    }

    // vraćanje podataka se može izvršiti u OnCreate ili u
    // OnRestoreInstanceState
    protected override void OnRestoreInstanceState(Bundle savedInstanceState)
    {
        base.OnRestoreInstanceState(savedInstanceState);
        brojac = savedInstanceState.GetInt("brojac", 0);
        ime_korisnika = savedInstanceState.GetString("ime_korisnika", "");
    }
}
```

Slika 3.7. Primer upotrebe metoda OnSavedInstanceState i OnRestoreInstanceState

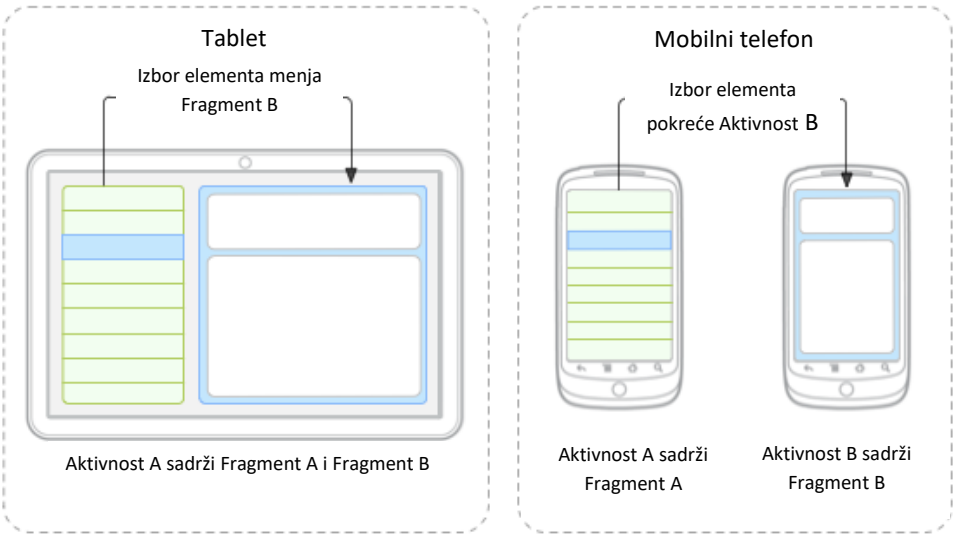
```
// this – instanca Aktivnosti iz koje se poziva nova Aktivnost
Intent intent = new Intent(this, typeof(ExampleActivity));
// prosledjivanje dodatnih podataka novoj Aktivnosti
intent.PutExtra("ime_korisnika", ime_korisnika);
StartActivity(intent);

// this – instanca Aktivnosti iz koje se poziva nova Aktivnost
Intent intentResult = new Intent(this, typeof(ExampleActivity));
// prosledjivanje dodatnih podataka novoj Aktivnosti
intent.PutExtra("ime_korisnika", ime_korisnika);
StartActivityForResult(intentResult, RESULT_CODE);
```

Slika 3.8. Primer poziva pokretanja nove Aktivnosti

3.3.2 Fragmenti

Fragmenti (*engl. Fragments*) su objekti koji predstavljaju delove korisničkog interfejsa, odnosno delove Aktivnosti. Ovi objekti su uvedeni u okviru verzije 3.0 operativnog sistema. Osnovna prednost upotrebe Fragmenta je povećanje modularnosti, fleksibilnost i ponovne upotrebe delova koda. Fragment zahteva Aktivnost da bi postojao, tako da je Fragmente moguće posmatrati kao modularne sekcije Aktivnosti. Jedna Aktivnost može sadržati više Fragmenata. Fragmenti su enkapsulirani, imaju svoju funkcionalnost, izgled i mogu reagovati na korisničke komande.



Slika 3.9. Prikaz upotrebe Fragmenta na tabletu i mobilnom telefonu

Fragmenti imaju svoj životni ciklus ali su zavisni od životnog ciklusa Aktivnosti u kom se nalaze. Tako da ukoliko Aktivnost pređe u stanje pauze, i sami Fragmenti koji se nalaze u okviru te Aktivnosti prelaze u stanje pauze. Njima je moguće manipulisati, odnosno po potrebi ih postavljati i uklanjati sa Aktivnosti. Dodavanje ili uklanjanje Fragmenta predstavlja jednu transakciju. Ove transakcije je moguće pamtiti na steku o kom se brine Aktivnost, čime se pruža mogućnost uklanjanja poslednje dodatog Fragmenta pritiskom na dugme „nazad“, čime se simulira opcija skidanja poslednje dodatog elementa na stek.

Na slici 3.10. je prikazan postupak definisanja novog Fragmenta sa postavljanjem željenog izgleda (datoteka šeme). Slika 3.11. daje primer postavljanja Fragmenta na Aktivnost i na stek. Životni ciklus Fragmenta je prikazan na slici 3.12.

```
public class ExampleFragment : Fragment
{
    View rootView;

    public override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
    }

    public override View OnCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState)
    {
      // postavljanje izgleda Fragmenta
      rootView = inflater.Inflate(Resource.Layout.example_fragment,
      container, false);

      // pretraga pogleda, dodeljivanje vrednosti

      return rootView;
    }
}
```

Slika 3.10. Primer definisanja Fragmenta

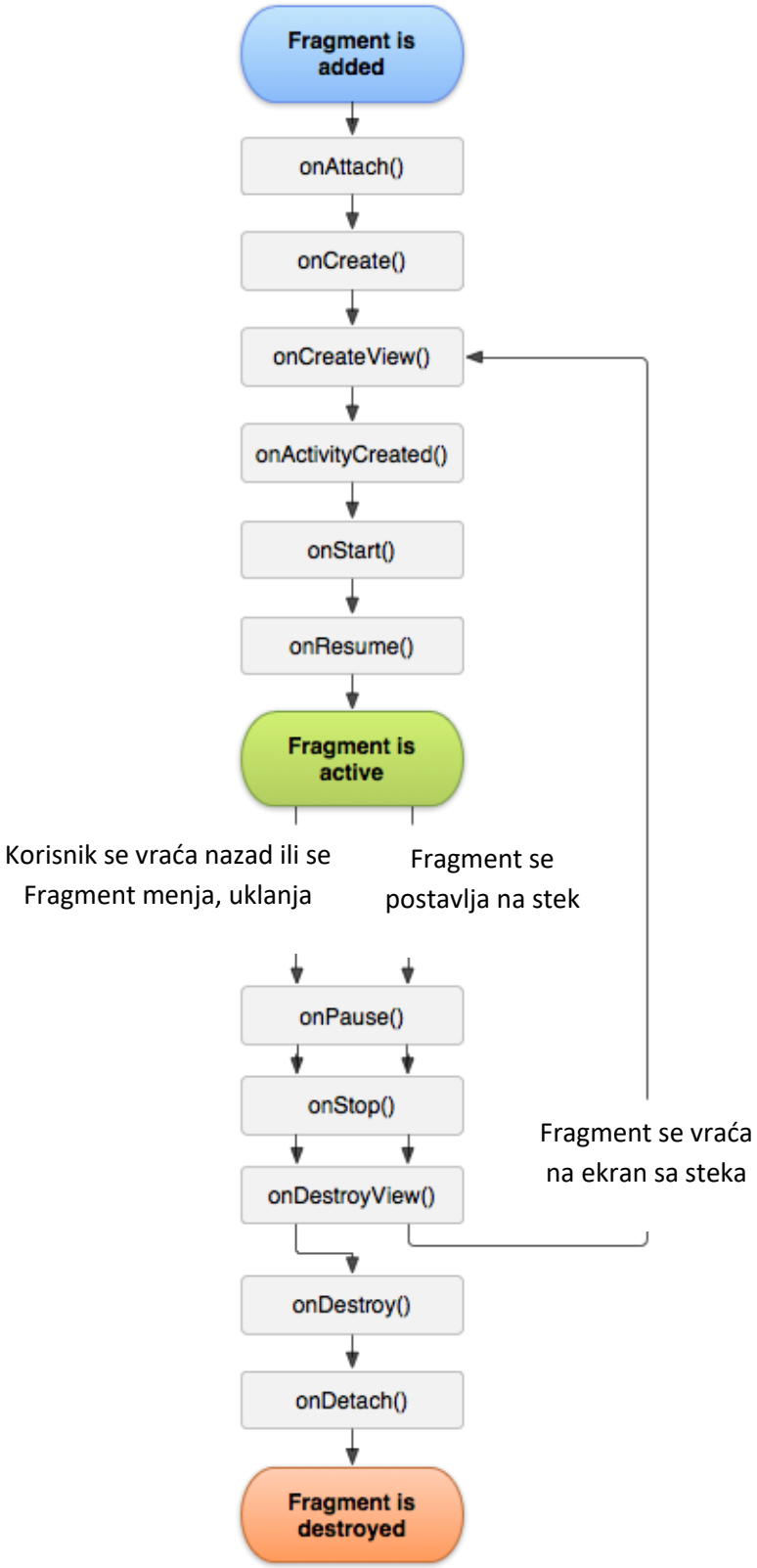
```
public class MainActivity : Activity
{

   protected override void OnCreate(Bundle savedInstanceState)
   {
      base.OnCreate(savedInstanceState);

      SetContentView(Resource.Layout.Main);
      // Container – objekat u koji se smesta Fragment
      var container = FindViewById<LinearLayout>(Resource.Id.container);
      // kreiranje novog Fragmenta, dodavanje u Activity i na stek
      ExampleFragment ef = new ExampleFragment();
      FragmentManager.BeginTransaction().Add(container.Id, ef, "example").
            AddToBackStack(null).Commit();

   }
}
```

Slika 3.11. Primer postavljanja Fragmenta na Aktivnost i dodavanje na stek



Slika 3.12. Životni ciklus Fragmenta

3.4 Android Manifest

Svaki projekat mora da sadrži datoteku *AndroidManifest.xml*, sa tačno tim nazivom. Ova datoteka pruža osnovne podatke o aplikaciji alatima za prevođenje i kompajliranje, operativnom sistemu i *Google Play* prodavnici. AndroidManifest se uglavnom kreira automatski pri kreiranju projekta i nalazi se u korenom direktorijumu. U toku razvoja ova datoteka se može menjati automatski ili po potrebi od strane programera. Primer jedne AndroidManifest datoteke dat je na slici 3.13.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.example.myapp">
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="26" />
    <application
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".DisplayMessageActivity"
            android:parentActivityName=".MainActivity" />
    </application>
</manifest>
```

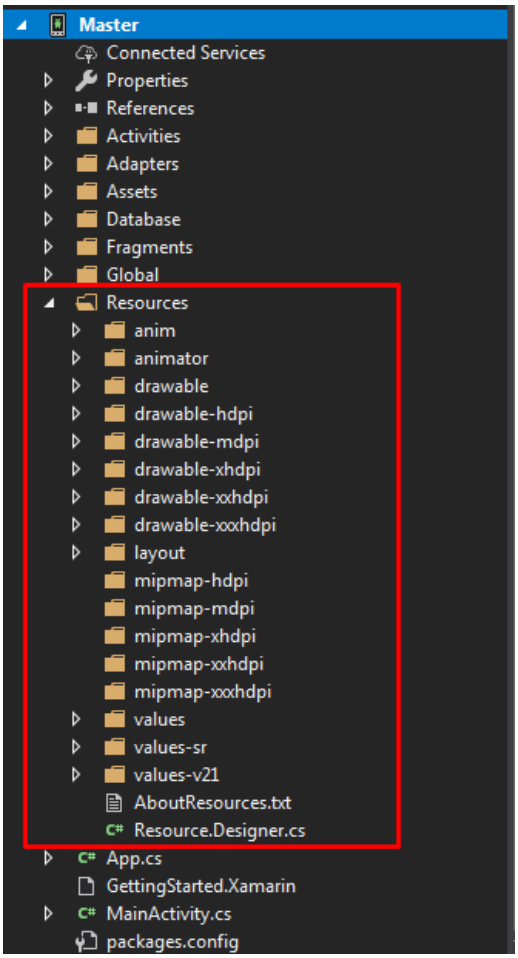
Slika 3.13. Primer AndroidManifest.xml datoteke

AndroidManifest, između ostalog, sadrži sledeće elemente:

- Ime aplikacije i ime paketa,
- Deklaraciju komponenti aplikacije: Aktivnosti, servise itd.
- Dozvole (*engl. Permissions*) potrebne za nesmetani rad aplikacije, kao što su: pristup WiFi ili *bluetooth* mreži, kontaktima, aplikaciji za pozive, lokaciji itd.
- Minimalna hardverska ili softverska specifikacija uređaja na kom se pokreće aplikacija.

3.5 Struktura projekta, *Resources* direktorijum

Svaka aplikacija koja se razvija za operativni sistem Android u okviru projekta mora sadržati direktorijum *Resources* (odnosno direktorijum *res* ukoliko se aplikacija razvija u jeziku Java). U okviru ovog direktorijuma nalaze se statički podaci kao što su niske, konstante, nizovi, slike, ikonice, fontovi, rasporedi odnosno šeme (*engl. Layout*) kao i datoteke generisane od strane razvojnog okruženja. Prikaz strukture projekta dat je na slici 3.14.



Slika 3.14. Struktura projekta sa naglašenim Resources direktorijumom

Resursne datoteke su nezavisne od ostatka koda i zbog toga se nalaze u zasebnom direktorijumu. Takođe, ovim putem olakšava se prilagođavanje izgleda aplikacije na ekranima različitih dimenzija, uvođenje više jezika aplikacije ili prilagođavanje različitim regijama (geografskim, kulturnim ili političkim). [4]

Neki direktorijumi u okviru *Resources* direktorijuma[2]:

- *Resources/anim* – sadrži definisane animacije kojima je moguće pristupiti iz koda
- *Resources/drawable* – sadrži grafičke elemente, najčešće slike (PNG, JPEG i dr.)
- *Resources/layout* – sadrži .xml datoteke u kojima je definisan raspored elemenata korisničkog interfejsa za različite elemente aplikacije
- *Resources/values* – sadrži niske, nizove, stilove i druge datoteke koje mogu biti referisane od strane drugih datoteka
- *Resources/raw* – sadrži opšte datoteke kao što su .CSV, audio datoteke i drugo
- *Resources/menu* – sadrži definicije elemenata menija koji se javljaju u aplikaciji.

Neki od navedenih direktorijuma se mogu pojaviti sa sufiksima. Tako direktorijumi *drawable-hdpi, drawable-mdpi, drawable-xhdpi, drawable-xxhdpi i drawable-xxxhdpi* sadrže grafičke elemente za ekrane različite rezolucije, a o izboru grafičkih elemenata koji će se koristiti u aplikaciji ne brine sam programer, već se određuje u toku pokretanja aplikacije.

Definisanjem više *layout* direktorijuma sa različitim sufiksima omogućava programerima da vrlo jednostavno definišu kako će korisnički interfejs izgledati na različitim uređajima. Tako da direktrorijum *layout-land* u sebi sadrži XML datoteke koje se koriste kada je uređaj u *landscape* (horizontalnom) režimu. Na raspolaganju je i specifičnije definisanje *layout* direktorijuma koje se vezuje za samu rezoluciju ekrana uređaja. Primer definisanja:

- *layout-w700dp* – za uređaje koji imaju širinu ekrana barem 700 dp
- *layout-h500dp* – za uređaje koji imaju visinu ekrana barem 500 dp

U okviru direktorijuma *values*, pored niski, nizova ili konstanti mogu se naći i *style.xml* datoteke koje definišu izgled (stil) kompletne aplikacije. U stilu mogu biti navedene primarne i sekundarne boje, definicije stilova pojedinačnih elemenata (dugmadi, padajućih listi itd.). Neki od mogućih sufiksa sa direktorijum *values* mogu biti:

- *values-v14* ili *values-v21*- gde v14 odnosno v21 predstavljaju verziju API biblioteka koje uređaj podržava, tako da aplikacija može da izgleda različito (kroz definiciju stilova) na uređajima sa KitKat ili Oreo verzijom sistema,

- *values-sr, values-en* itd. – omogućava korišćenje više jezika u aplikaciji u odnosu na regiju (geografsku, političku ili kulturnu).

Resursne datoteke je moguće koristiti iz koda ili u okviru drugih resursnih datoteka. Primer dobijanja resursnih datoteka dat je na slici 3.15.

```
// dobijanje slike iz Resource direktorijuma
Drawable drawable = Activity.GetDrawable(Activity, Resource.Drawable.slika);
// dobijanje niske iz niski definisanih u Resource/values/Strings
string niska = Activity.GetString(Resource.String.app_name);
```

Slika 3.15. Primer dobijanja resursnih datoteka

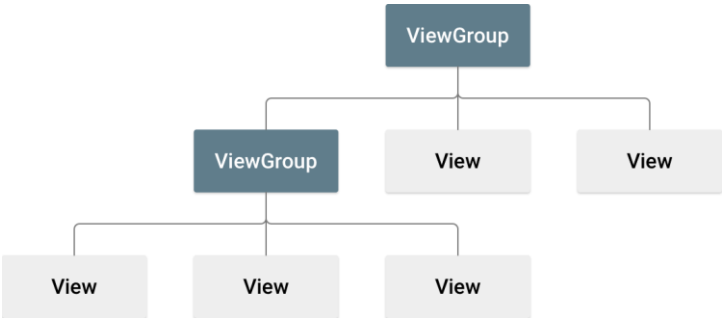
3.6 Pogledi, klasa *View*

Osnovni gradivni element korisničkog interfejsa u operativnom sistemu Android je pogled (*engl. View*). Pogled zauzima pravougaonu oblast na ekranu uređaja i odgovoran je za iscrtavanje elementa i reagovanje na korisničke komande u toj oblasti.

Klasa *View* predstavlja baznu klasu za sve poglede koje korisnik može videti i koji mogu imati interakciju sa korisnikom (*Button, CheckBox, SwitchButton itd.*). Ovi pogledi se nazivaju vidžeti (*engl. Widgets*).

Klasa *ViewGroup* predstavlja baznu klasu za poglede koji načešće nisu vidljivi korisniku i služe kao kontejneri za više drugih objekata tipa *View* ili *ViewGroup*. Primeri klasa koje nasleđuju *ViewGroup* su *LinearLayout, RelativeLayout, FrameLayout* itd.[3]

Svi pogledi i grupe pogleda u jednom prozoru se predstavljaju pomoću strukture stabla. Novi pogledi se u strukturu dodaju kao potomci već postojećih pogleda. Struktura jednog takvog stabla data je na slici 3.16.



Slika 3.16. Struktura hijerarhije pogleda predstavljena pomoću stabla

Kao što je već objašnjeno, raspored i međusobni odnos elementa korisničkog interfejsa mogu biti definisani u okviru XML datoteka ili u samom kodu. Resursne datoteke šeme sadrže poziciju elemenata, njihovu veličinu, atribute i funkcionalnost (akcije). Praksa je da se inicijalni raspored elemenata definiše u okviru XML datoteka, dok se izmene koje se dešavaju u toku izvršavanja definišu u kodu. Ovom praksom se omogućava ponovna upotreba datoteka šema jer su nezavisne od koda. Ujedno se ostavlja prostor za fleksibilnost i prilagođavanje situaciji u kojoj se ta šema koristi, odnosno omogućava se promena ponašanja i funkcionalnosti same šeme.

Resurne datoteke šema se definišu pomoću XML jezika, analogno kao kod HTML jezika za definisanje veb stranica. Svaka šema mora sadržati jedan koreni (*engl. Root*) element koji mora biti tipa *View* ili *ViewGroup*, a zatim se ostali elementi mogu dodavati kao potomci (ugnježđeni) u stablu. Tako formatirane datoteke se smeštaju u ditektorijum *Resource/layout* pod ekstenzijom .xml. Primer definicije jedne datoteke šeme pod nazivom *test.xml* dat je na slici 3.17. dok je na slici 3.18. prikazana definicija pogleda u kodu.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical" >
     <TextView android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a TextView" />
     <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Slika 3.17. Primer definicije datoteke šeme test.xml

```
LinearLayout container;
TextView textView = new TextView(Activity);
textView.Text = "Hello World!";
textView.SetTextColor(Android.Graphics.Color.Green);
container.AddView(textView);
```

Slika 3.18. Primer definicije pogleda u kodu

Svaki pogled, odnosno grupa pogleda, ima svoje parametre i atribute. Pomoću atributa definišu se dimenzije pogleda, senke, boja, tekst, relativna pozicija, jedinstveni identifikator itd. Svi atributi mogu biti definisani u XML datoteci ili u kodu.

U korenom elementu navodi se imenski prostor koji omogućava korišćenje atributa iz tog imenskog prostora. Na primer:

- `xmlns:android="http://schemas.android.com/apk/res/android"`
- `xmlns:app="http://schemas.android.com/apk/res-auto"`
- `xmlns:tools="http://schemas.android.com/tools"`

Neki od parametara koji se koriste pri definisanju pogleda:

`android:id` – jedinstveni identifikator preko kog je moguće pronaći određeni pogled u stablu pogleda.

`android:layout_width, android:layout_height` – postavljanje visine i širine pogleda. Vrednosti koje ovaj parametar uzima mogu biti izraženi pomoću:

- tačnih dimenzija – broj piksela, dp, sp.
- `wrap_content` – tačno koliko je potrebno da obuhvati element sa svojim sadržajem,
- `match_parent` – maksimalno koliko je moguće da zauzme prostora u roditeljskom pogledu.

`android:padding` – prazan prostor oko sadržaja unutar elementa, inicijalno je 0. Na raspolaganju su i `paddingLeft, paddingRight, paddingTop i paddingBottom.`

`android:layout_margin` – prazan prostor oko elementa, inicijalno je nula. Na raspolaganju su i `marginLeft, marginRight, marginTop i marginBottom.`

`android:layout_weight` – proporcijalni odnos dimenzija dva ili više pogleda. Odnosno koliko prostora pogled može da zauzme u odnosu na druge poglede izraženo kao proporcija.

`android:elevation` – koliko je pogled izdignut po Z osi u odnosu na druge poglede.

`android:gravity` – određuje pozicioniranje objekata u okviru pogleda. Može imati vrednosti *top, bottom, left, right, center, center_vertical, center_horizontal*.

`android:layout_gravity` – određuje kako će se element pozicionirati u roditeljskom pogledu.

`android:visibility` – određuje vidljivost elementa. Može imati vrednosti *visible, invisible i gone*, gde *invisible* znači da se element ne vidi ali zauzima prostor na ekranu predviđen za taj pogled, dok *gone* znači totalno sklanjanje pogleda sa prikaza.

`android:background` – boja ili tekstura pozadine elementa.

`android:orientation` – kako će pogledi biti slagani u okviru elementa, horizontalno ili vertikalno (najčešće u okviru *LinearLayout* elementa).

Svaki pogled takođe sadrži i atribut *Tag* koji može biti bilo koji objekat i najčešće se koristi za pamćenje podataka vezanih za taj pogled. Detaljnije će biti predstavljena upotreba u narednom poglavlju.

Za pretragu pogleda u stablu na raspolaganju su sledeće funkcije, dok će primer upotrebe ovih funkcija biće dat na slici 3.20:

- `FindViewById(int id)` – vraća objekat tipa View koji ima navedeni ID.
- `FindViewWithTag(Object tag)` – vraća objekat View koji ima navedeni Tag.

```
public class ExampleFragment : Fragment
{
   public override void OnCreate(Bundle savedInstanceState)
   {
      base.OnCreate(savedInstanceState);
   }

   public override View OnCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState)
   {
      // postavljanje korenog elementa stabla pogleda
      View v = inflater.Inflate(Resource.Layout.example, container, false);
      // pretraga sa kastovanjem pomocu FindViewById
      Button btn1 = v.FindViewById<Button>(Resource.Id.btn1);
      Button btn2 = (Button)v.FindViewById(Resource.Id.btn2);
      // pretraga sa kastovanjem pomocu FindViewWithTag
      Button btn3 = v.FindViewWithTag<Button>("dugme");
      Button btn4 = (Button)v.FindViewWithTag("dugme2");

      return v;
   }
}
```

Slika 3.20. Primer pretrage pogleda u okviru Fragmenta

Osnovni pogledi koji nasleđuju *ViewGroup*:

LinearLayout – najčešće korišćen pogled za ostvarivanje uloge kontejnera. Elementi u njemu se slažu linearno jedan ispod drugog ili jedan iznad drugog u zavisnosti od atributa *orientation*.

RelativeLayout – elementi se u okviru ovog pogleda pozicioniraju u odnosu na druge elemente. Pruža veliku fleksibilnost i može dovesti do smanjenja broja ugnježdenih pogleda korišćenjem funkcija za određivanje pozicija kao što su:

- `android:layout_alignParentTop` – pozicionira pogled potomak ispod gornje ivice roditeljskog pogleda (postoje i verzije za *Bottom, Left i Right*)
- `android:layout_below`– pozicionira pogled ispod pogleda čiji je ID naveden kao vrednost ovog parametra (postoje i verzija sa *layout_above*)
- `android:layout_centerVertical` – pozicionira pogled tako da bude centriran vertikalno (postoji i verzija *layout_centerHorizontal*)
- `android:layout_toRightOf` – pozicionira pogled desno od pogleda čiji je ID naveden kao vrednost ovog parametra (postoji i verzija *layout_toLeftOf, layout_toEndOf, layout_toStartOf*)

FrameLayout – elementi se u okviru ovog pogleda postavljaju jedan iznad drugog po principu steka, odnosno preklapaju se. Najčešće se koristi sa jednim direktnim pogledom potomkom.

Neki pogledi koji nasleđuju klasu *View* (vidžeti):

Button – objekat tipa dugme, najčešće se uparuje sa funkcijom koja se izvršava kada se dugme pritisne. Može sadržati tekst i ikonicu (postavljanjem atributa *drawableLeft, drawableRight, drawableTop* ili *drawableBottom*).

TextView – objekat za prikaz teksta.

EditText – objekat za unos teksta. Može određivati da li se prikazuje numerička ili slovna tastatura preko parametra *inputType*.

ImageView – objekat za prikaz slika.

Switch, CheckBox – objekti slični dugmadima koji samo mogu imati stanje markiran ili nemarkiran, odnosno uključen ili isključen.

Spinner – objekat koji predstavlja padajuću listu. Kada se ne prikazuje padajuća lista, ispisuje trenutno odabranu vrednost. Koristi se zajedno sa:

- *SpinnerAdapter* klasom koja popunjava elemente padajuće liste i
- *AdapterView.OnItemSelectedListener* funkcijom koja reaguje na odabir elementa iz padajuće liste

ListView – skrolabilan objekat koji služi sa prikazivanje liste elemenata. Svaki element liste može imati svoju šemu, mada je praksa da svi elementi koriste istu šemu. Koristi se zajedno sa *ListAdapter-om* koji vodi računa o podacima ri izgleda svakog reda liste. Funkcija *GetView* koja popunjava pogled svakog reda liste se poziva svaki put kada novi red postane vidljiv.

RecyclerView – novija i naprednija verzija prikaza liste podataka. Uvodi nove optimizacije i ponovno upotrebljivanje istih pogleda, otuda naziv RecyclerView. Koristi se sa RecyclerView.Adapter-om i RecyclerView.LayoutManager-om koji vode računa o iscrtavanju novih redova u listi, promenama u podacima, animacijama pri prikazu novih redova.

ViewPager – objekat grafičkog interfejsa koji omogućava horizontalno straničenje. Najbolji primer upotrebe ovog objekta je u aplikaciji galerije, gde se slike mogu „listati“ na levo i na desno pokretom prevlačenja. Omogućava olakšanu organizaciju sadržaja koji može biti grupisan u celine. Veoma često se ovaj element koristi zajedno sa prikazom tabova koji mogu sadržati ikonice i tekst. Ima veliku primenu u segmentu za podešavanja ili kao linija sa menijem (izborom strana). Primer upotrebe *ViewPager* elementa zajedno sa tabovima će biti prikazan u poglavlju implementacije aplikacije „Učigraonica“.

Preporuke za optimizaciju i rad sa listama podataka biće predstavljene u narednom poglavlju.

4 Saveti, pravila, preporuke, obrasci

Pri dizajniranju korisničkog interfejsa na operativnom sistemu Android, ali i dizajniranju interfejsa za bilo koju platformu, postoje načela, pravila, saveti, dobre prakse i preporuke za postizanje dobro definisanog i funkcionalnog okruženja.

4.1 Razumevanje korisnika, vizuelne naznake i nagrađivanje korisnika

Jedan od osnovnih principa je razumevanje korisnika kojima je aplikacija namenjena, kao i sama svrha aplikacije. Upotreba aplikacije se može značajno razlikovati među korisnicima raznog životnog doba, zanimanja, predznanja kao i mnogih drugih okolnosti koje manje ili više utiču na način korišćenja. Učešćem samih korisnika pri razvoju aplikacije smanjuje mogućnost stvaranja „neupotrebljivog“ rezultata ili nezadovoljstva i otpornosti krajnih korisnika. Razumevanje korisničkih navika je aspekt koji često bude zanemaren a može značajnu uticati na nezadovoljstvo korisnika. Princip:

„*Ne teraj me da razmišljam*!“ (*engl. Don’t make me think!*)

govori da korisnik ne bi trebalo da provede značajno vreme razmišljajući kako da koristi aplikaciju. Odnosno da prvo korisničko pitanje kada ugleda aplikaciju ne bi smelo da bude „Šta da uradim sledeće?”. [1]

Ovaj princip se primenjuje na sve slojeve i elemente korisničkog interfejsa. Standardna lokacija i raspored navigacionih tastera, položaj menija, lako prepoznatljive ikonice samo su neke od stvari o kojima treba voditi računa.

Veoma bitnu stavku predstavljaju „vizuelne naznake“ elemenata korisničkog interfejsa. Ove naznake bi trebalo korisniku da šalju kratku poruku kako se određeni element korisiti pa čak i koji će ishod interakcije sa tim elementom biti. Tako na primer ikonica na kojoj se nalazi kućica govori korisniku da se radi o povratku na naslovnu stranu aplikacije, dok ikonica kante za đubre najčešće signalizira brisanje. Korišćenjem pogrešnih ili teško prepoznatljivih vizuelnih naznaka može zbuniti, iznervirati ili oštetiti korisnike. Ljudi dosta sporije primećuju i pamte tekst nego slike, što je dosta pogodno pri razvoju aplikacija za mobilne uređaje gde je prostor od velikog značaja. Interfejs

mora pružati odgovor na svaku akciju korisnika. Ne smeju se događati situacije da je korisnik preduzeo određenu akciju a da ni sam nije siguran da li je interfejs odreagovao, i da li je njegova akcija zabeležena. U svrhu obaveštavanja korisnika dobra praksa je korišćenje objekata *Toast* i *Snack* koji izbacuju kratke poruke, u malom „oblaku“, na ekranu korisnika ne prekidajući tok izvršavanja aplikacije i ne ometajući korisničke akcije.

Princip koji je sve više dobija na značaju je takozvani sistem nagrađivanja korisnika (*engl. Gamification*). Ovaj sistem je osnovno načelo preuzeo iz video igara koji predstavljaju sve jaču industriju. U ljudskoj prirodi je da teži ka takmičenju, nadmetanju i postizanju novih ciljeva. Poenta je nagraditi korisnika za rezultat ili trivijalne radnje koje obavlja u aplikaciji. Nagrade su najčešće jednostavna obaveštenja da je postignut određeni rezultat, ali mogu biti i pozicioniranje na rang listi najboljih rezultata, otključavanje pristupa posebnom delu aplikacije ili nagrade virtualnim novcem, tokenima, trofejima. Akcije koje pružaju nagrade mogu biti jednostavne, od svakodnevnog ulaska u aplikaciju do postizanja dobrih rezultata. Ovaj sistem u mnogome ubrzava proces učenja upotrebe aplikacije ali i tera korisnika da češće koristi aplikaciju. [1]

4.2 Konzistentnost i ekrani aplikacije

Aplikaciju iz ugla korisničkog interfejsa treba posmatrati kao niz ekrana koji su međusobno povezani i korisnik se može kretati kroz različite ekrane. Bitan aspekt korisničkog interfejsa je konzistentnost, odnosno doslednost. Ukoliko su ekrani funkcionalno povezani, raspored elemenata na tim ekranima bi trebalo da bude sličan. Ovim se postiže da korisnik već zna gde se određeni elementi nalaze i da se pomoću dosadašnjeg iskustva vrlo brzo i jednostavno može snaći u novom ekranu. Doslednost se može primenjivati i na boje, font, animacije, izbor slika i ikonica kao i sam način interakcije sa interfejsom radi postizanja određenog cilja (da li se od korisnika očekuje dodir, duži dodir, pokret ili neki drugi način akcije). Tako da ukoliko se u jednom delu aplikacije korisnik kroz listu kreće jednostavnim povlačenjem liste na gore (skrolovanjem), i ostale liste bi trebalo da slede isti sistem upotrebe, a ne da se recimo kroz listu kreće pritiskom na dugme. Takođe, ukoliko korisnik koristi aplikaciju na više

različitih uređaja, iako aplikacije može izgledati drugačije, način upotrebe bi trebalo da bude intuitivan i već poznat.

Posmatranjem aplikacije kao niza ekrana olakšava planiranje i osmišljavanje izgleda tih ekrana. Crtanjem korisničkog interfejsa uz pomoć papira i olovke ili upotrebom softvera za pravljenje prototipova ekrana interfejsa dovodi do bržeg uočavanja greški i nelogičnosti, a ujedno dovodi i do lakšeg uvođenja izmena i uključivanja korisnika u razvoj aplikacije već pri osmišljavanju interfejsa.

4.3 Upotreba biblioteka

Velika prednost razvijanja aplikacija za Android operativni sistem je jaka zajednica programera na Internetu. Pored direktne pomoći zajednice kroz odgovore na pitanja i primere koda, najveća prednost je veliki broj već razvijenih biblioteka, od kojih je većina besplatna.

Android biblioteka je strukturno identična modulu Android aplikacije. Biblioteka može da sadrži sve potrebene komponente (resurse, izvorni kod, Android manifest, slike itd.) koje su potrebne za kreiranje APK datoteke. Android biblioteka se prevodi u AAR datatoteku (*engl. Android Archive*) umesto u APK. Biblioteke su izrazito korisne u sledećim situacijama [10]:

- Ukoliko se razvija više aplikacije koje koriste iste module, aktivnost, elemente korisničkog interfejsa, servise itd.
- Ukoliko se razvija više verzija jedne iste aplikacije (npr. besplatna i plaćena varijanta) a koje koriste iste osnovne komponente

Upotrebom biblioteka podiže se modularnost i ponovna upotreba koda. Samim tim ukoliko je u toku razvoja aplikacije potreban neki određeni element, skup animacija, biblioteka za rad sa slikama itd. postoji velika šansa da je već razvijena biblioteka za tu namenu. Biblioteke razvijane od strane zajednice treba korisiti pažljivo, ali otvaraju dosta novih mogućnosti, ubrzavaju proces razvoja i mogu značajno unaprediti aplikaciju. Nezavisno od jezika u kom se aplikacija razvija (Java ili C#) moguće je učitati i korisiti sve AAR biblioteke.

U nastavku će biti predstavljene samo neke biblioteke koje mogu biti od koristi pri razvoju korisničkog interfejsa:

Picasso – je besplatna Android biblioteka za rad sa slikama. U okviru ove veoma male ali funkcionalne biblioteke nalaze se alati za:

- Brzu i jednostavnu obradu (transformaciju) slika, promenu rezolucije, dimenzija, upotrebu efekata uz minimalnu upotrebu memorije
- Rukvanje sa *ImageView* elementima koristeći optimizaciju kroz ponovnu upotrebu sadržaja
- Preuzimanje slika sa mreže ili učitavanje iz lokalne memorije, uz mogućnost kontrole da li se slike preuzimaju upotrebom WiFi ili mobilne mreže
- Automatsko keširanje slika u radnoj memoriji ili na internoj memoriji uređaja

TransitionsEverywhere – je Android biblioteka koja omogućava olakšan rad sa animacijama. TransitionsEverywhere je korišćena pri razvoju aplikacije "Učigraonica" čija će implemetacija biti opisana u sledećem poglavlju. Funkcionalnost ove biblioteke je ogromna i značajno može uticati na izgled ali i funkcionalnost aplikacije. Kao što je već opisano animacije igraju značajnu ulogu u korisničkom iskustvu pri upotrebi aplikacije. Razvijanje sopstvenih animacija može biti zahtevno i može oduzimati dosta vremena, što je osnovni moto za razvoj ove biblioteke. Upotreba biblioteke je izrazito jednostavna uz veoma malo koda. Uz pomoć samo par linija koda pri kreiranju ekrana ili pogleda postižu se automatske animacije za sve promene koje se dešavaju u tim pogledima. Tako će promena teksta, sakrivanje elemenata korisničkog interfejsa, promena boje ili veličine, promena položaja ali i mnogi drugi događaji imati prateće animacije.

FButton – je Android biblioteka koja omogućava rad sa posebno dizajniranim 3D dugmadima koja imaju animaciju pri pritisku na dugme. Ova biblioteka je takođe korišćena prilikom razvoja aplikacije "Učigraonica".

Zoomage – je Android biblioteka koja implementira mogućnost uveličavanja slika (*engl. Pinch-to-zoom*) koristeći *ImageView* objekat.

Pored ovih biblioteka, neke od često korišćenih biblioteka su:

- *Transitioner* – biblioteka za rad sa animacijama
- *FragmentRigger* – biblioteka za rad sa Fragmentima

- *GSON (Newton)* – biblioteka za rad sa JSON formatom (serijalizacija i deserijalizacija objekata)
- *AnimatedPieView* – biblioteka za rad sa grafikonima (pitama) kao načinom prikaza podataka
- *Syncfusion* – komercijalna bilioteka koja sadrži ogroman broj alata za prikaz podataka, liste, *GridView*, tabelarni prikaz itd.

4.4 Stilovi i teme

Čest je slučaj da je potrebno da se određeni skup atributa primeni na više elemenata kako bi ti elementi izgledali slično ili identično. Taj skup atributa može biti vezan za pogled, Fragment, Aktivnost pa čak i može biti primenjen na celu aplikaciju. Na primer, sva dugmad u aplikaciji bi trebalo da imaju istu animaciju, dimenzije, boju, aplikacija ne treba da sadrži akcioni meni, font u celoj aplikaciji treba da bude identičan. U slučaju upotrebe stila na nivou cele aplikacije to nazivamo tema (*engl. Theme*). Sve ovo se može postići definisanjem sopstvenih stilova, koji se kasnije mogu primeniti na željeni pogled, Aktivnost ili celu aplikaciju. Znači, uloga stilova je da izvrši enkapsulaciju grupe atributa koji se mogu upotrebiti po potrebi, više puta nezavisno od datoteka šema. [2]

Ukoliko u aplikaciji postoje pogledi koji imaju zajedničke atribute i izgledaju slično dobra praksa je te atribute izdvojiti u stil da bi se izbeglo ponavljanje i kopiranje koda.

Stilovi se definišu u okviru *Resources/values* direktorijuma kao XML datoteke, dok se u kodu mogu koristiti uz navođenje `style="@style/ime_stila"`. To je sve što je potrebno za upotrebu stilova, bez ikakvih izmena u Android Manifest datoteci ili bilo gde drugo u kodu. Svaka datoteka stila se sastoji od taga `<style>` u kom se navodi ime stila i "parent" ukoliko taj stil treba da nasledi već postojeći stil i ugnježdenih tagova `<item>` u kojima se definišu željeni atributi i vrednosti tih atributa. Primer definisanja jednog stila dat je na slici 4.1.

Postavljanje stila kao teme se postiže navođenjem stila u Android Manifest datoteci. Primer definisanja teme i izmene u Manifest-u dat je na slikama 4.2. i 4.3.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="MojStil" parent="TextAppearance.AppCompat">
    <item name="android:textColor">#FFFF00</item>
    <item name="android:textSize">13sp</item>
  </style>
</resources>
```

Slika 4.1. Definisanje stila

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorSecondary</item>
  </style>
</resources>
```

Slika 4.2. Definisanje teme aplikacije

```xml
<manifest ... >
  <application ... >
    <activity android:theme="@style/AppTheme" ...="" >
    </activity>
  </application>
</manifest>
```

Slika 4.3. Postavljanje teme aplikacije u Android Manifest datoteci

4.5 Definisanje sopstvenih animacija

Animacije igraju značajnu ulogu pri razvoju korisničkog interfejsa. U Android operativnom sistemu animacije se mogu korisiti na mnogim mestima počevši od prelaska iz jedne Aktivnosti u drugu, prikazivanja novog sadržaja ili Fragmenta, promene osobina nekog objekta itd. Veliki broj predefinisanih animacija koje su na raspolaganju programerima postoji u API bibliotekama. Međutim, veoma često je potrebno da se definišu animacije koje su specifične i nisu predefinisane.

Takve animacije je moguće definisati u okviru *Resources/anim* direktorijuma kao XML datoteke. Na raspolaganju su dve vrste animacija.

Slika-po-slika animacija koja prikazuje slike navedene u okviru `<animation-list>` taga u navedenom vremenskom intervalu. Primer definicije jedne takve animacije

(srceAnimacija.xml) sa potrebnim slikama (slika 4.4.) dat je na slici 4.5. zajedno sa kodom potrebnim za korišćenje definisanih animacija na slici 4.6.

Slika 4.4. Potrebene slike za definisanje slika-po-slika animacije

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/srce1" android:duration="200" />
  <item android:drawable="@drawable/srce2" android:duration="200" />
  <item android:drawable="@drawable/srce3" android:duration="200" />
  <item android:drawable="@drawable/srce4" android:duration="200" />
  <item android:drawable="@drawable/srce5" android:duration="200" />
</animation-list>
```

Slika 4.5. Primer definisanje slika-po-slika animacije (srceAnimacija.xml)

```
ImageView im = (ImageView)FindViewById(Resource.Id.imageView);
im.SetBackgroundResource(Resource.Animation.srceAnimacija);
AnimationDrawable ad = (AnimationDrawable)im.Background;
ad.Start();
```

Slika 4.6. Primer upotrebe definisane animacije u okviru ImageView pogleda

Drugi tip animacija su animacije koje vrše transformacije nad jednom slikom ili objektom. Transformacije koje je moguće primeniti su [3]:

- *AlphaAnimation* – kontroliše providnost
- *RotateAnimation* – kontroliše rotacije
- *ScaleAnimation* – kontroliše promene u veličini (visine i širine)
- *TranslateAnimation* – kontroliše poziciju

Ove animacije je moguće koristiti odvojeno, navedene u tagovima `<alpha>`, `<rotate>`, `<scale>` i `<translate>`, ili kombinovano u okviru `<set>` taga.

Atributi za `<alpha>`:

`android:fromAlpha, android:toAlpha–` stepen providnosti (može uzeti vrednosti od 0.0 do 1.0)

Atributi za `<rotate>`:

`android:fromDegrees, android:toDegrees` – ugao koji rotacija obuhvata

`android:pivotX, android:pivotY` – tačka oko koje se vrši rotacija

Atributi za `<scale>`:

`android:fromXScale, android:toXScale` – veličina transformacije po X osi

`android:fromYScale, android:toYScale` – veličina transformacije po Y osi

`android:pivotX, android:pivotY` – tačka koja će biti fiksirana prilikom transformacije

Atributi za `<translate>`:

`android:fromXDelta, android:toXDelta` – pomeranje po X osi

`android:fromYDelta, android:toYDelta` – pomeranje po Y osi

Primer definisanja jedne takve animacije dat je na slici 4.7. kao i kod potreban za upotrebu animacije na slici 4.8.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <alpha
      android:duration="1000"
      android:fromAlpha="0.0"
      android:toAlpha="1.0" />
  <rotate
      android:duration="1000"
      android:fromDegrees="90"
      android:toDegrees="180" />
</set>
```

Slika 4.7. Primer definisanja animacije koja menja providnost objekta i rotira ga za 90 stepeni

```
TextView textView = (TextView)FindViewById(Resource.Id.textView);
Animation animFadein = AnimationUtils.LoadAnimation(Activity,
Resource.Anim.fadeIn);
textView.StartAnimation(animFadeIn);
```

Slika 4.8. Menjanje providnosti objekta tipa TextView

4.6 Shared Animations

Tranzicija između različitih Aktivnosti ili Fragmenata najčešće se obavlja jednostavnom animacijom kompletnog ekrana ili dela ekrana. Međutim postoji način da element ili više elemenata iz jedne Aktivnosti "pređe" u drugu Aktivnost i tako korisniku olakša praćenje toka aplikacije i upotpuni korisničko iskustvo. Najjednostaviji primer tog ponašanja je aplikacija galerije na Androidu gde se pritiskom na sliku iz liste slika, ta ista slika, počevši od polaznog mesta, poveća i raširi tako da zauzme ceo ekran ali sve u jednoj glatkoj animaciji. Time se obezbeđuje da korisnik više pažnje obraća na sam sadržaj i postiže se bolje i fluidnije korisničko iskustvo. Elementi koji su zajednički za dve Aktivnosti ili dva Fragmenta nazivaju se *Shared Elements*, dok se ceo postupak prelaska naziva *Shared Elements Transition*.

Shared Elements Transition je mehanizam koji automatski izračunava početni i završni položaj i dimenzije zajedničkih elemenata i bez intervencije programera postavlja animaciju koja će se desiti u trenutku prelaska. Mehanizam je jednostavan za upotrebu i fleksibilan je. Dozvoljava izbor animacije, takođe ako sadržaj koji je zajednički zahteva vreme za učitavanje moguće je odlaganje animacije dok sadržaj ne bude spreman za prikaz.

U nastavku će biti predstavljen postupak postizanja objašnjenog efekta na primeru dve Aktivnosti koje imaju zajednički element *ImageView*.

Prvi korak je postavljanje polja `WindowContentTransitions` na `true` što se može postići u datoteci stila (prikazano na slici 4.9.) ili u kodu Aktivnosti.

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
  <item name="android:windowContentTransitions">true</item>
  ...
</style>
```

Slika 4.9. Postavljanje polja WindowContentTransitions na true

Drugi korak je dodeljivanje atributa `TransitionName` pogledima koji su zajednički. Ovo je moguće uraditi u XML datoteci šeme ili u kodu. Na slici 4.10. prikazan je postupak dodeljivanja ovog atributa elementu tipa *ImageView*.

```
<ImageView
    android:id="@+id/imageView"
    android:transitionName="zajednicki_element"
    android:layout_width="100dp"
    android:layout_height="160dp" />
```

Slika 4.10. Postavljanje polja TransitionName u elementu tipa ImageView

Veoma je bitno da oba elementa imaju istu vrednost ovog polja, element u početnoj Aktivnosti kao i element u završnoj Aktivnosti.

Treći korak je pokretanje nove Aktivnosti preko funkcije `StartActivity(Intent intent, Bundle options)`. Primer je dat na slici 4.11.

```
ImageView imageView = (ImageView)FindViewById(Resource.Id.imageView);
Intent intent = new Intent(this, typeof(SecondActivity));
// Ukoliko treba proslediti dodatne podatke
intent.PutExtra("dodatni_podaci", data);
// prvi parametar – Aktivnost iz kog se poziva
// drugi parametar – zajednički element
// treći parametar – TransitionName koji je korišćen
ActivityOptionsCompat options = ActivityOptionsCompat.
MakeSceneTransitionAnimation(this, imageView, " zajednicki_element ");
StartActivity(intent, options.ToBundle());
```

Slika 4.11. Primer poziva završne Aktivnosti (SecontActivity) iz početne Aktivnosti (FirstActivity)

Da bi se animacija zajedničkih elemenata izvršila i po izlasku iz završne Aktivnosti potrebno je pozvati metod `FinishAfterTransition()` umesto `Finish()`.

Ukoliko se zajednički element iz završne Aktivnosti oslanja na asinhrono učitavanje sadržaja i potrebno je odložiti animaciju dok se učitavanje ne završi, potrebno je pozvati metodu `PostponeEnterTransition()`. Tek kada sadržaj sigurno bude učitan animacija se pokreće pozivom metode `StartPostponedEnterTransition()`.

Još jedna mogućnost ovog mehanizma je postavljanje više zajedničkih elemenata koji će biti animirani pri prelasku u novu Aktivnost. Na primer da Aktivnosti sadrže istu sliku, naslov i tekst ispod slike, sva tri elementa je potrebno dodati u parametar pri pozivu `MakeSceneTransitionAnimation`. Primer je dat na slici 4.12.

```
ImageView imageView = (ImageView)FindViewById(Resource.Id.imageView);
TextView naslov = (TextView)FindViewById(Resource.Id.naslov);
TextView text = (TextView)FindViewById(Resource.Id.text);
Pair<View, String> p1 = Pair.Create((View)imageView, "trans1");
Pair<View, String> p2 = Pair.Create((View)naslov, "trans2");
Pair<View, String> p3 = Pair.Create((View)text, "trans3");
ActivityOptionsCompat options = ActivityOptionsCompat.
MakeSceneTransitionAnimation(this, p1, p2, p3);
StartActivity(intent, options.ToBundle());
```

Slika 4.12. Primer poziva ukoliko postoji više zajedničkih elemenata

Postupak SharedElementsTransition u okviru Fragmenta je korišćen pri razvoju aplikacije "Učigraonica" i biće predstavljen u narednom poglavlju.

Takođe moguće je fiksirati statusnu liniju i akcioni meni pri prelasku iz jedne Aktivnosti u drugu, čime se postiže efekat promene samo sadržaja umesto celog ekrana.

4.7 Osvežavanje pogleda iz drugih niti

Uređaji koji pokreću Android operativni sitem uglavnom imaju procesore sa više niti (*engl. Thread*) što omogućava programerima pokretanje novih niti u toku aplikacije koje mogu izvršavati neke manje ili više zahtevne zadatke. Iako operativni sistem može imati na raspolaganju veći broj niti uvek je samo jedna nit zadužena za iscrtavanje, prikaz i osvežavanje korisničkog interfejsa, u nastavku UI nit (*engl. User Interface Thread*). Ukoliko ta nit bude preopterećena, ne bude odzivna na 5 sekundi, sistem će izbaciti prozor sa obaveštenjem da je aplikacije postala neodzivna i pitanje da li korisnik želi da zatvori trenutnu aplikaciju. Da bi se izbegao navedeni scenario dobra praksa je da za zahtevne operacije programer pokrene zasebnu nit. Problem nastaje ukoliko nova nit ima potrebu da nakon izvršenog posla osveži pogled ili više pogleda na UI niti. Komunikacija između niti se može obaviti na više načina, međutim u nastavku će biti predstavljen najčešći slučaj koji koristi poziv metode *RunOnUiThread*. Dovoljno je pozvati ovu metodu iz nove niti i u okviru lambda operatora izvršiti željene izmene. Na slici 4.13. je dat primer upotrebe *RunOnUiThread* gde nekoliko pogleda osvežava svoje stanje.

```
if (Activity != null)
    Activity.RunOnUiThread(() =>
    {
        textView.Text = "Novi tekst!";
        imageView.SetImageResource(Resource.Drawable.slika1);
        checkBox.Checked = true;
    });
```

Slika 4.13. Primer upotrebe metode RunOnUiThread za osvežavanje pogleda

4.8 Rad sa listama podataka, *ViewHolder* obrazac

Liste su veoma čest način prikazivanja podataka na uređajima sa Android operativnim sistemom. Prvenstveno zbog ograničenja u prostoru ekrana, podatke je pogodno smeštati jedan ispod drugog i jednostavnim pokretom prevlačenja se kretati kroz listu. *ListView* je jedan od objekata koji može poslužiti za prikaz podataka u listi. Primere liste ekrana moguće je naći na mnogo mesta, od prikaza memorisanih kontakta, spiska instaliranih aplikacija, SMS poruka, liste poziva itd.

Osnovna karakteristika liste je da pruži dovoljno informacija o podacima koje predstavlja, ali da ne bude previše komplikovana i teška za praćenje ili pretragu. Tako jedan red liste ne bi trebalo da sadrži više od nekoliko informacija, npr. u slučaju prikaza kontakta to je najčešće samo slika ili ikonica sa imenom vezanim za taj kontakt. Daljom navigacijom, odnosno pritiskom na red liste, ukoliko je to potrebno, može se obezbediti detaljniji prikaz o podatku iz tog reda, u slučaju kontakta to bi bili brojevi telefona, e-mail adresa, kontrole za izmenu itd.

Druga karakteristika liste bi trebalo da bude brzina, odnosno fluidnost. Odzivnost i brzina kretanja kroz listu u mnogome utiču na iskustvo korisnika pri korišćenju segmenta aplikacije sa listama podataka.

U operativnom sistemu Android sve potrebne funkcije za prikaz reda se izvršavaju tek kada novi red u listi treba prikazati na ekranu. Ovaj princip otvara mogućnosti za optimizaciju brzine liste. Optimizacije liste podataka su mnogobrojne, ali se zasnivaju na količini podataka koji moraju da se obrade pri kretanju kroz listu i broju poziva pretrage i povezivanja pogleda sa tim podacima.

Broj podataka koje treba obraditi pri inicijalizaciji novog reda liste se može smanjiti uporšćavanjem prikaza jednog reda liste, odvajanjem zahtevnije obrade podataka u zasebnu nit ili obezbeđivanjem podataka unapred.

Međutim, pretraga pogleda i povezivanje sa podacima se izvršava svaki put kada novi red treba prikazati na ekranu. Ovo se može izbeći upotrebom obrasca *ViewHolder,* koji će biti inicijalizovan samo onoliko puta koliko u jednom trenutku može biti redova liste na ekranu. ViewHolder sadrži sve potrebene poglede za prikaz jednog reda, i time se broj poziva funkcija za pretragu pogleda značajno smanjuje. Svaki pojedinačni *ViewHolder* se puni podacima samo jedanput, pri inicijalizaciji. Tako da ukoliko lista koju treba prikazati ima 1000 redova sa po tri pogleda u svakom redu, za prolazak kroz celu listu potrebno je 3000 poziva metode *FindViewById*, jer se za svaki red ta metoda poziva tri puta. Dok upotrebom obrasca *ViewHolder*, ukoliko je na ekranu moguće prikazati 8 redova, metoda *FindViewById* će biti pozvana samo 24 puta, odnosno 3 puta za svaki od 8 *ViewHolder-a*.

Kao što je već navedeno, klasa koja se bavi obradom pojedinačnih redova liste naziva se *adapter*. Adapter sadrži metodu *GetView*, koja se poziva svaki put kada novi red treba prikazati na ekranu, a svaki red ima svoj koreni pogled, odnosno koren stabla pogleda za taj red. *ViewHolder* će biti inicijalizovan u ovoj metodi i biće smešten u polje *tag* korenog pogleda reda. U okviru polja *tag* moguće je smestiti bilo koji objekat. Da bi se izbegla višestruka inicijalizacija *ViewHolder*-a prvo se proverava da li je polje *tag* već popunjeno postojećim *ViewHolder*-om, ukoliko jeste, uzima se taj objekat, a ukoliko nije kreira se novi *ViewHolder*, popunjavaju se njegova polja i smešta se u polje *tag*.

Na slici 4.16. dat je primer kompletne klase adaptera sa implementiranim obrascem *ViewHolder*. Svaki red liste ima tri pogleda, jednu sliku i dva tekstualna polja. XML datoteka koja je korišćena za prikaz jednog reda, data je na slici 4.14. pod nazivom *list_element_layout.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical"
      android:gravity="center"
      android:padding="5dp" >
      <ImageView
            android:id="@+id/slika"
            android:layout_width="50dp"
            android:layout_height="50dp" />
      <TextView
            android:id="@+id/text1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
      <TextView
            android:id="@+id/text2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
</LinearLayout>
```

Slika 4.14. Definicija XML datoteke list_element_layout.xml

Slika 4.15. Prikaz jednog reda liste

Ukoliko je potrebno da neki pogled zapamti dodatni podatak, taj podatak se može smestiti u polje T*ag*. Ovo polje može biti veoma značajno ukoliko se koristi u listi podataka. Može se desiti da redovi liste imaju pogled kao što je dugme, padajuća lista, *checkbox* ili bilo koji drugi vidžet koji može prihvatati korisničke akcije, onda se u polje *tag* može smestiti podatak o broju reda, element niza objekata koji je prikazan u tom redu, ali i mnogi drugi.

Iako postoji noviji objekat *RecyclerView* koji implementira ViewHolder obrazac zajedno sa dodatnim optimizacijama, *ListView* je i dalje veoma često korišćen pogled zbog svoje fleksibilnosti, brže implementacije događaja pri korisničkoj interakciji, odnosno dodirom na sam red liste, ili zbog potrebe da se redovi liste razlikuju u odnosu na neki faktor.

```
public class ViewHolder : Java.Lang.Object
{
    public TextView Naziv { get; set; }
    public TextView Podnaziv { get; set; }
    public ImageView Slika { get; set; }
}

public class ListAdapter : BaseAdapter<Film>
{
    List<Film> filmovi;
    Activity context;
    public ListAdapter(Activity pContext, List<Film> pItems) : base()
    {
        this.context = context;
        this.filmovi = pItems;
    }
    public override long GetItemId(int position)
    {
        return position;
    }

    public override int Count
    {
        get { return filmovi.Count; }
    }

    public override Film this[int position]
    {
        get { return filmovi[position]};
    }

    public override View GetView(int position, View convertView, ViewGroup
parent)
    {
      View view = convertView;
      Film lFilm = filmovi[position];
      ViewHolder holder = convertView.Tag as ViewHolder;
      if (holder == null)
      {
            view =
context.LayoutInflater.Inflate(Resource.Layout.list_element_layout, null);
            holder = new ViewHolder();
            holder.Slika = view.FindViewById<ImageView>(Resource.Id.slika);
            holder.Naziv = view.FindViewById<TextView>(Resource.Id.text1);
            holder.Podnaziv = view.FindViewById<TextView>(Resource.Id.text2);
      }
      // osvezavanje pogleda
      holder.Naziv.Text = lFilm.Naziv;
            ...

      view.Tag = holder;
      return view;
  }
}
```

Slika 4.16. Implementacija obrasca ViewHolder u adapteru koji radi sa klasom Film

5 Implementacija aplikacije „Učigraonica”

U ovom poglavlju će biti predstavljena implementacija aplikacije „Učigraonica“ koja koristi veliki broj principa programiranja korisničkog interfejsa na operativnom sistemu Android, a koji su navedeni u okviru teorijskog dela ovog rada. U okviru prikaza implementacije neće biti navođen kompletan izvorni kod aplikacije, već samo neki ključni delovi. Za razvoj aplikacije korišćen je jezik C# i razvojno okruženje *Visual Studio 2017* i *Xamarin.Android*. Aplikacija je otvorenog koda i on se može naći na adresi: `https://github.com/panucci/Ucigraonica-Master`

5.1 Opis aplikacije

Aplikacija „Učigraonica“ je namenjena deci uzrasta od 6-12 godina i koncipirana je kao igra kroz koju deca mogu naučiti osnovne matematičke operacije (sabiranje, oduzimanje, množenje i deljenje), kao i da kroz pogađanje pojmova sa slike nauče abecedu ili azbuku, a samim tim lakše savladaju čitanje i pisanje. Aplikacija takođe čuva i najbolje rezultate iz svih vrsta igre, čime se u praksi prikazuje upotreba „gejmifikacije“, omogućava podešavanje težine zadataka iz matematike kroz izbor maksimalne vrednosti za zbir ili razliku. Kompletna aplikacija se može koristiti u dva pisma, latinici ili ćirilici. Aplikacija omogućava tri različite igre: matematika, matematika sa ponuđenim odgovorima i čitanje i pisanje.

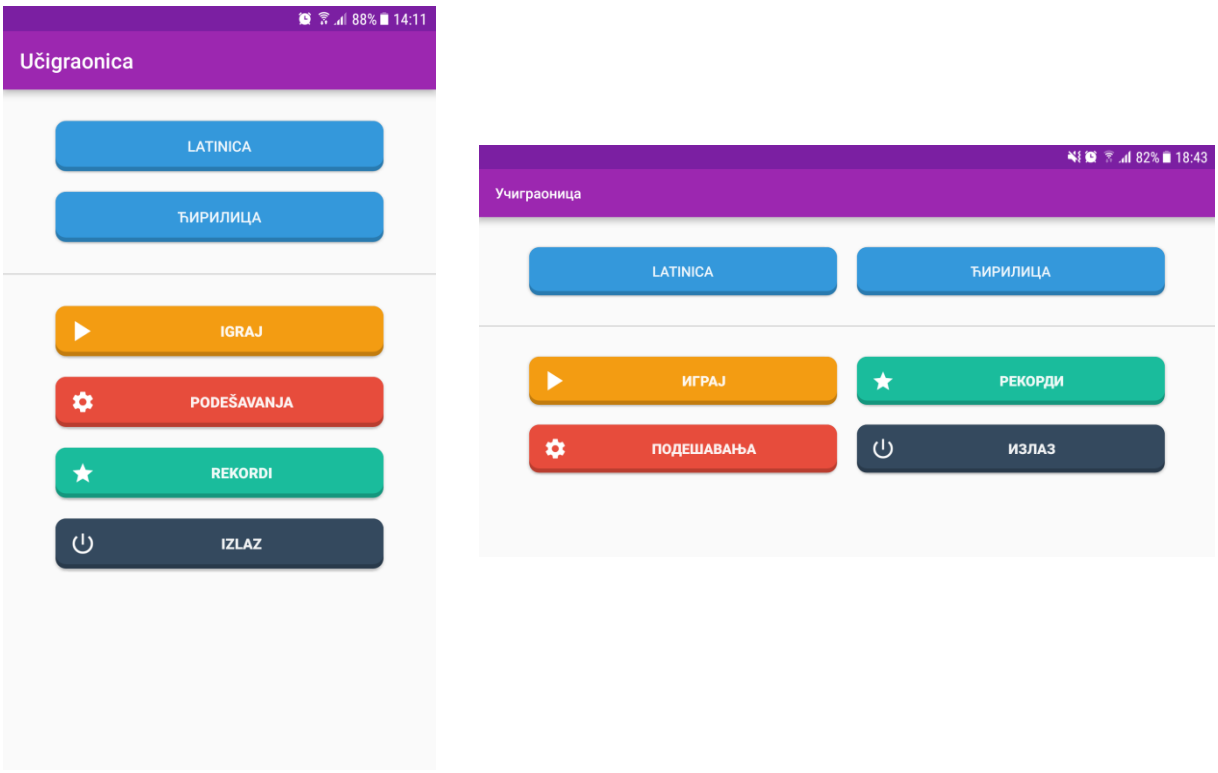
Igra matematika zahteva od igrača da pogodi rezultat traženog zadatka unoseći preko numeričke tastature svoj odgovor. Moguć je izbor četiri osnovne matematičke operacije. Pred igračem je 10 zadata koje mora da reši da bi se upisao na listu najboljih rezultata. Nije moguće preći na sledeći zadatak dok se ne reši trenutni. Igrači se rangiraju po vremenu potrebnom za rešavanje svih 10 zadataka. Nakon rešenih dobijenih zadataka, igrač ima mogućnost ponovne igre ili povratka u sekciju za izbor igre. Lista najboljih rezultata se odvojeno pamti za sve matematičke operacije.

Igra matematika sa odgovorima zahteva od igrača da odabere jedan od četiri ponuđena odgovora od kojih samo jedan može biti tačan. Moguć je izbor četiri osnovne matematičke operacije. Kao i u igri matematika, pred igračem je 10 zadataka. Igra vodi računa samo o pogođenim zadacima. Nakon završenih svih 10 zadataka prikazuje se

broj pogođenih odgovora i vreme potrebno za rešavanje svih zadataka. Rangiranje igrača na listi najboljih rezultata se vrši prvo po broju pogođenih odgovora a zatim po vremenu koje je bilo potrebno. Nakon rešenih dobijenih zadataka, igrač ima mogućnost ponovne igre, ili povratka u sekciju za izbor igre. Lista najboljih rezultata se odvojeno pamti za sve matematičke operacije.

Igra čitanje i pisanje zahteva od igrača da pogodi pojam sa slike unoseći svoj odgovor preko prikazane tastature na ekranu. Na raspolaganju je 195 pojmova koje je moguće pogađati koristeći ćirilicu ili latinicu. Pojmovi su podeljeni u tri novoa, laki, srednji i teški. U aplikaciji je težina predstavljena brojem zvezdica koji se osvajaju kada se pojam pogodi, 1 zvezdica za lake pojmove, 2 za srednje i 3 za teške. Aplikacija sabira broj osvojenih zvezdica i prikazuje ih kao zbir u listi rezultata, odvojeno za ćirilicu i latinicu.

Pri pokretanju aplikacije prikazuje se početni ekran na kom je moguće izabrati pismo aplikacije (ćirilica ili latinica), što automatski menja sve reči i niske u aplikaciji u odabrano pismo. Pored izbora pisma na početnom ekranu aplikacije se nalaze i opcije „Igraj“, „Podešavanja“, „Rekordi“ i „Izlaz“. Početni ekran ima različit raspored dugmadi u zavisnosti od položaja ekrana (vertikalnli ili horizontalni). Slika 5.1. prikaz početnog ekrana aplikacije.



Slika 5.1. Početni ekran aplikacije u vertikalnom i horizontalnom režimu

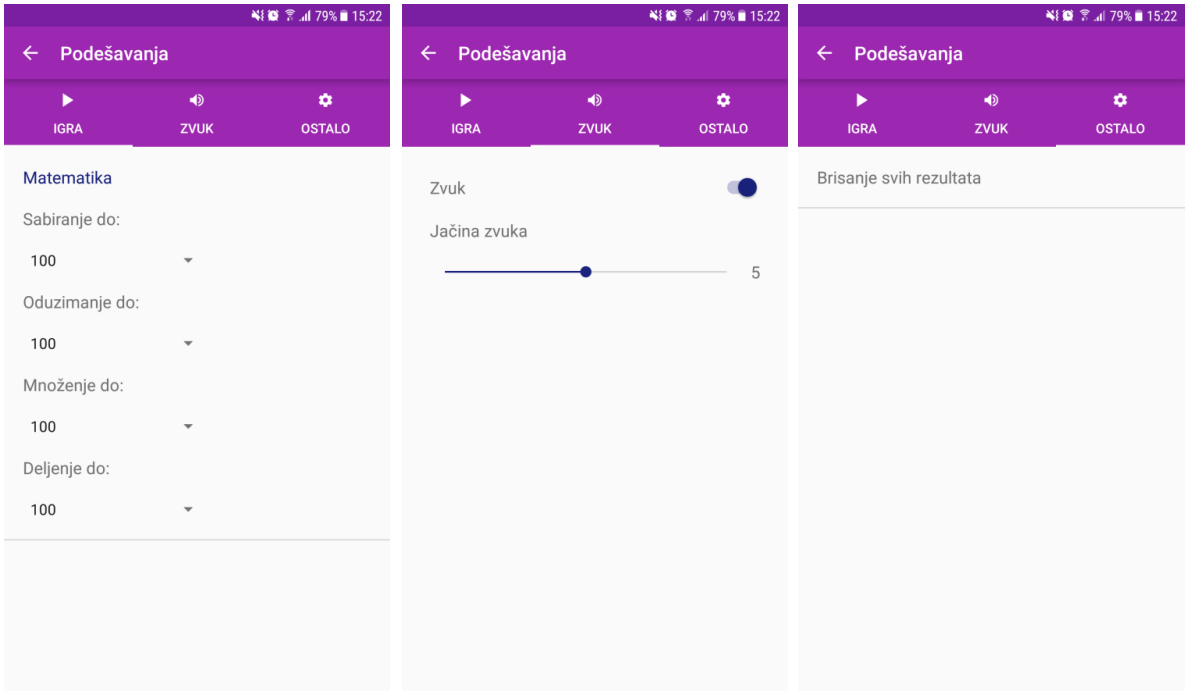
Odabirom opcije „Igraj“ ulazi se u segment aplikacije za izbor igre. Na raspolaganju su tri opcije „Čitanje i pisanje“, „Matematika sa ponuđenim odgovorima“ i „Matematika“ koje predstavljaju tri opisane igre. Uz igre u kojima se uči matematika, nalazi se i dugme za izbor matematičke operacije. Na slici 5.2. je dat prikaz ekrana sa izborom igre.

Slika 5.2. Izbor igre

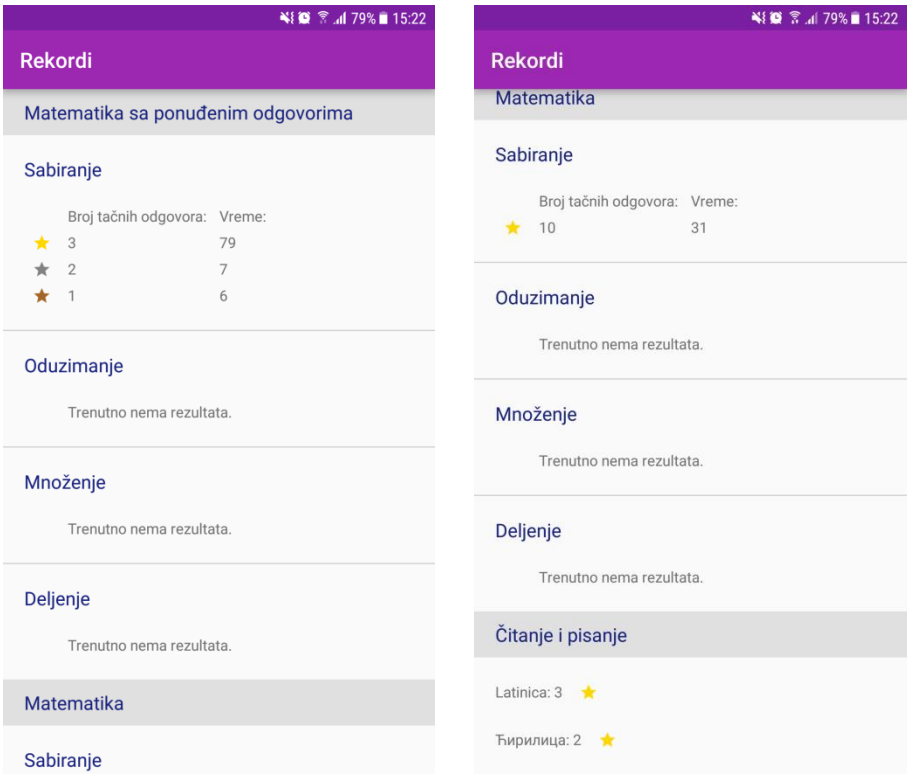
Odabirom opcije „Podešavanja“ ulazi se u segment aplikacije sa opcijama za težinu matematičkih igara. Za sabiranje podešava se maksimalna vrednost zbira, za oduzimanje maksimalna vrednost umanjenika, za množenje maksimalna vrednost proizvoda, a za deljenje maksimalna vrednost deljenika. Takođe, ovde se nalaze podešavanja jačine zvuka i opcija za brisanje liste rezultata. Na slici 5.3. je dat prikaz svih sekcija sa podešavanjima.

Odabirom opcije „Rekordi“ ulazi se u segment aplikacije gde se nalaze liste najboljih ostvarenih rezultata matematičkih igara i broja sakupljenih zvezdica u igri čitanja i pisanja. Rezultati matematičkih igara su podeljeni u kategorije matematičkih operacija za obe igre posebno. Prikazuju se tri najbolja rezultata ostvarena, sortirani po broju pogođenih zadataka i utrošenom vremenu u sekundama. Na slici 5.4. je dat prikaz ekrana sa rekordima.

Odabirom opcije „Izlaz“ izlazi se iz aplikacije. Onemogućeno je izaći iz aplikacije pritiskom na dugme „nazad“ da igrač greškom ne bi mogao da izađe iz aplikacije.



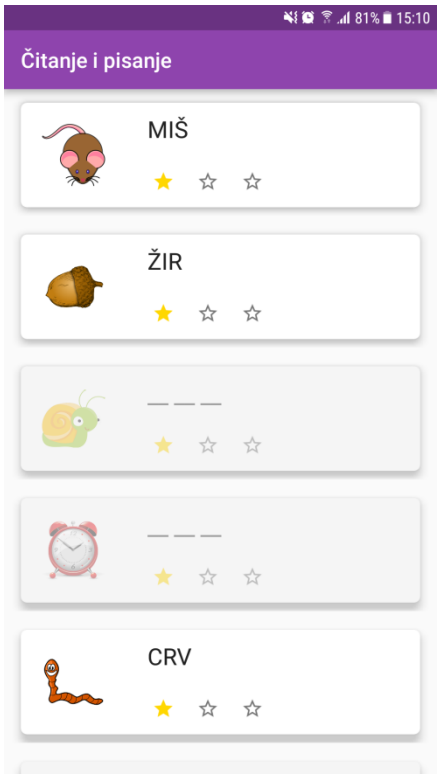
Slika 5.3. Podešavanja



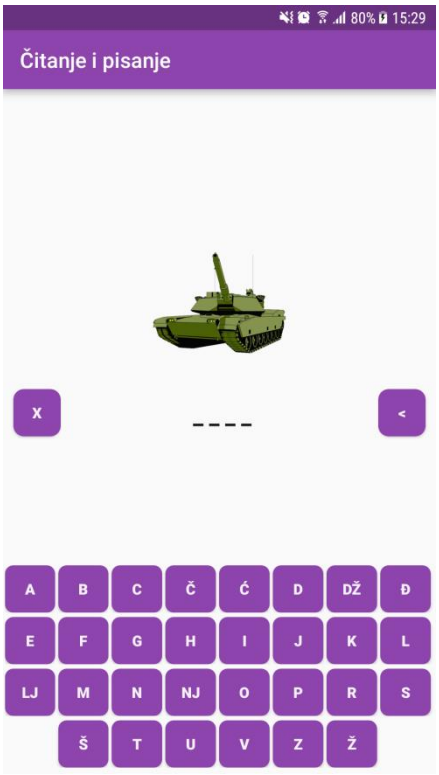
Slika 5.4. Rekordi

Kao što je već prikazano, postoje tri različite igre, sa različitim korisničkim interfejsom za svaku igru posebno, i različitim interfejsom za vertikalni i horizontalni prikaz. Takođe svaka igra ima „svoju“ boju koju prate i elementi korisničkog interfejsa vezanih za tu igru.

Igra čitanja i pisanja, nakon odabira opcije „Čitanje i pisanje“, prikazuje listu pojmova koje treba pogoditi (Slika 5.5). Već pogođeni pojmovi će u samom redu liste prikazivati i pogođeno rešenje, dok će još nerešeni pojmovi biti blago zatamnjeni i rešenje neće biti prikazano. Dodirom na bilo koji pojam otvara se novi prozor u kom igrač može uneti svoj odgovor preko date tastature. Ukoliko igrač unese tačan odgovor, automatski se prikazuje sledeći pojam. (Slika 5.6)



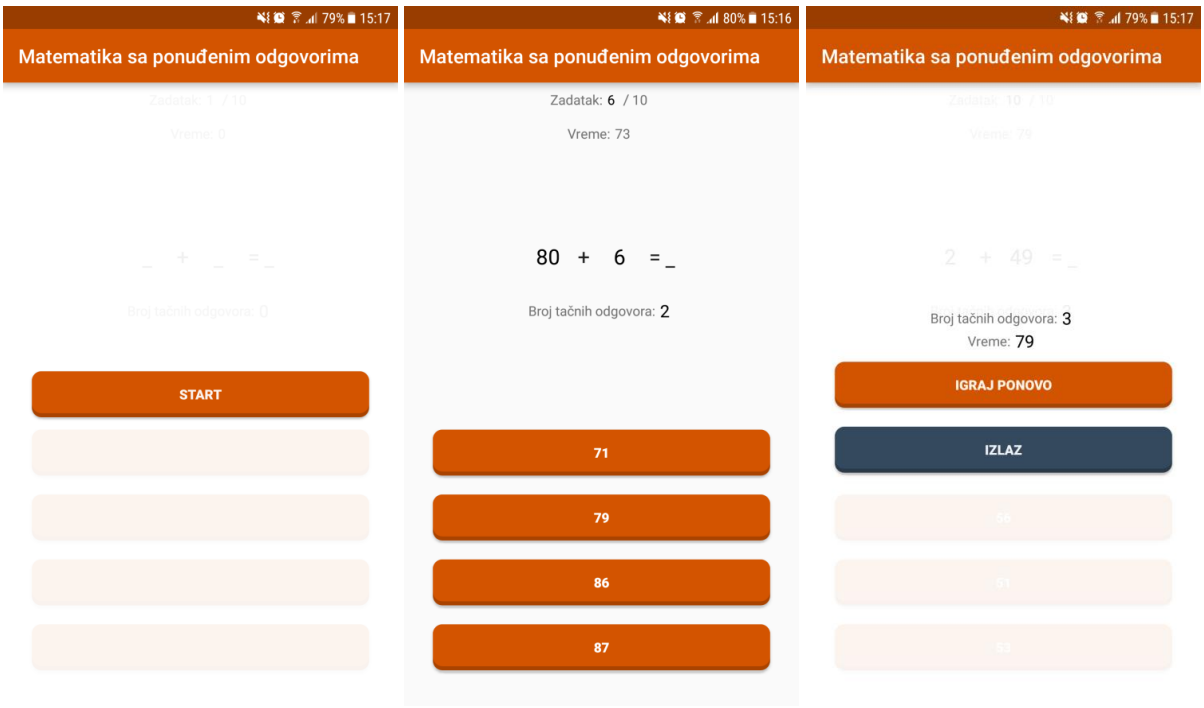
Slika 5.5. Lista pojmova



Slika 5.6. Pogađanje pojmova

Igra matematika sa ponuđenim odgovorima, nakon odabira opcije „Matematika sa ponuđenim odgovorima“ i izborom željene operacije, otvara novi prozor u kom se nalazi dugme „Start“ za pokretanje jedne runde od 10 zadataka (slika 5.7.). Nakon toga prikazuje se redni broj zadatka, vreme koje je prošlo, sam zadatak, broj tačnih odgovora i četiri ponuđena odgovora. Odmah pri izboru odgovora oblast oko tekst zadatka će zasvetleti na kratko zeleno ili crveno u zavisnosti da li je odgovor tačan ili ne, time se

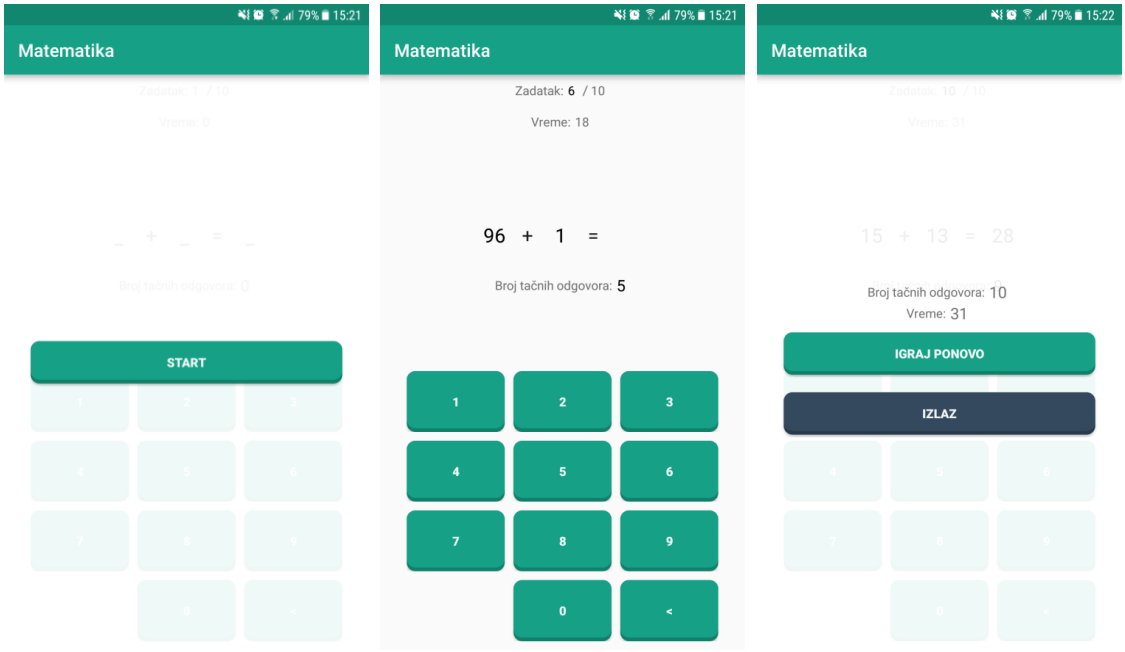
postiže da korisnik odmah ima reakciju igre na njegov odgovor. Nakon odgovaranja na svih 10 zadataka nudi se opcija nove igre ili povratka u segment izbora igre.



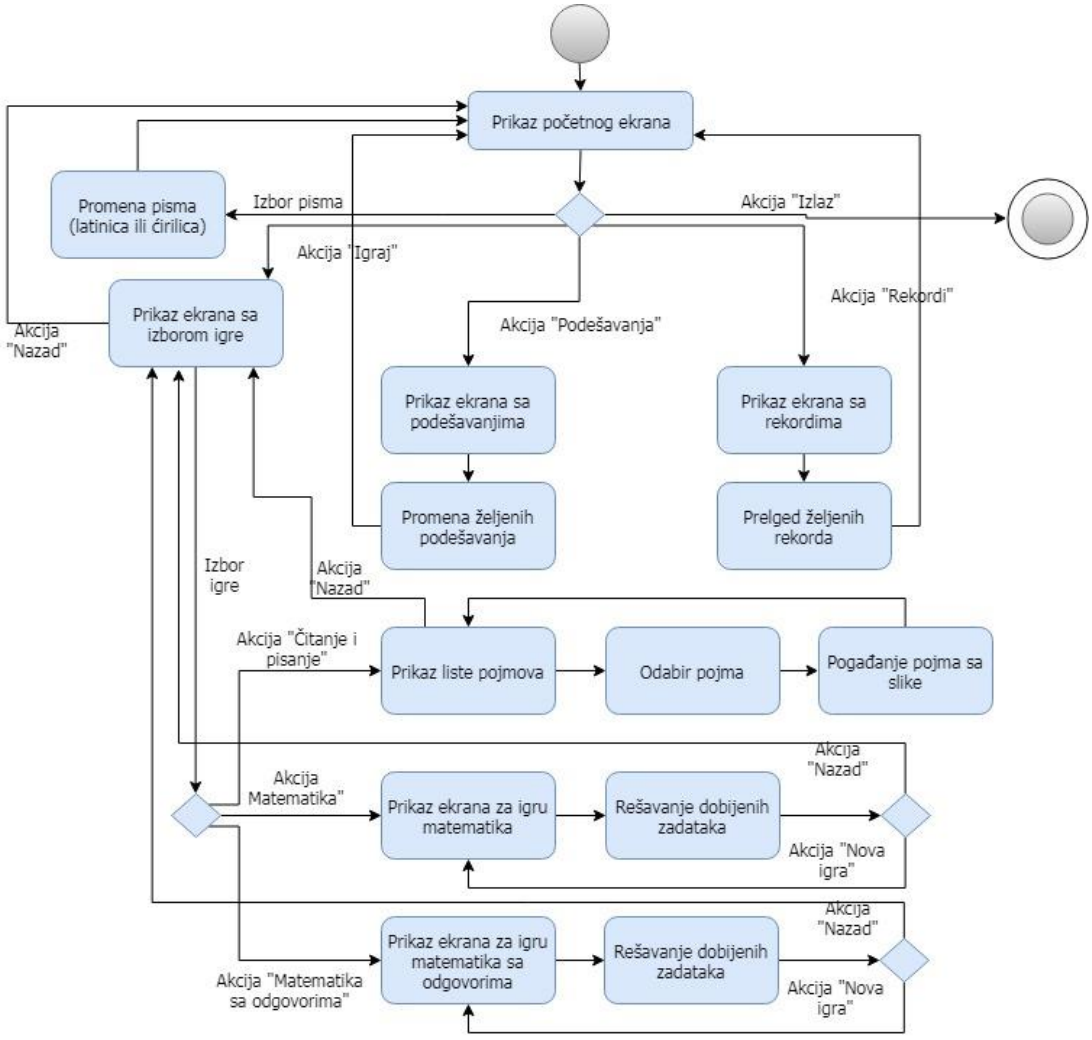
Slika 5.7. Igra matematika sa odgovorima

Igra matematika, nakon odabire opcije „Matematika“ i izborom željene operacije, otvara novi prozor u kom se nalazi dugme „Start“ za pokretanje jedne runde od 10 zadataka. Nakon toga prikazuje se redni broj zadatka, vreme koje je prošlo, sam zadatak, broj tačnih odgovora i numerička tastatura za unos odgovora. Nakon odgovaranja na svih 10 zadataka nudi se opcija nove igre ili povratka u segment izbora igre. Prikaz jedne runde igre „Matematika“ dat je na slici 5.8.

Na slici 5.9. dat je dijagram aktivnosti gde je prikazan način upotrebe aplikacije kroz niz mogućih akcija.



Slika 5.8. Igra matematika



Slika 5.9. Dijagram aktivnosti

5.2 Klasa Application

Klasa *Application* je osnovna klasa u okviru Android aplikacije koja sadrži sve ostale komponente kao što su Aktivnosti i servisi. Klasa *Application*, ili njena podklasa, se instancira pre svih drugih klasa u trenutku kada se proces za samu aplikaciju kreira. Najčešća upotreba ove klase je za inicijalizaciju globalnog stanja aplikacije, odnosno za čuvanje podataka koji su dostupni na nivou cele aplikacije. Pored toga, ova klasa se može korisiti za izvršavanje metoda koje je neophodno izvršiti pre prikazivanja prve Aktivnosti.

Ukoliko postoji potreba za uvođenje sopstvene klase *Application*, to je moguće postići nasleđivanjem klase *Android.App.Application* i specifikacijom u *AndroidManifest* datoteci.

U okviru aplikacije „Učigraonica“ uvedena je klasa *App* koja nasleđuje *Android.App.Application* i čija je namena čuvanje podataka o bazi podataka, podešavanjima aplikacije, izboru jezika, lista slova iz azbuke i abecede kao i drugih statičkih polja. Ovim je pristup bazi podataka moguć iz bilo kog dela koda preko same klase *App* i njenog polja *db*. Definisanje klase *App* prikazano je na slici 5.10. dok je na slici 5.11. prikazana sama *AndroidManifest* datoteka.

```
public class App : Application
{
	public static CDatabase db;
	public static Preferences preferences;
	public static App Current;
	public static Locale default_locale = Locale.Default;
	public static LinearLayout fragmentContainer;
	public static Fragment CurrentFragment;
	public static string MathCode = "m2";
	public static string MathWithAnswersCode = "m1";
	public static int CurrentImagePossition = -1;

	public static string[] Abeceda;
	public static string[] Azbuka;

	public App(IntPtr handle, global::Android.Runtime.JniHandleOwnership
transfer)
		: base(handle, transfer)
	{
		Current = this;
		preferences = new Preferences();
		Abeceda = new string[] { "A", "B", "C", "Č","Ć", "D","Dž",
"Đ","E", "F","G", "H","I", "J","K", "L", "Lj", "M", "N", "Nj","O", "P","R",
"S","Š", "T","U", "V","Z", "Ž" };

		Azbuka = new string[] { "А", "Б", "В", "Г","Д", "Ђ", "е","Ж",
"З","И", "Ј","К", "Л","Љ", "М","Н", "Њ","О", "П", "Р", "С","Т", "Ћ","У",
"Ф","Х", "Ц","Ч", "Џ","Ш"};
		ReadSharedPreferences();
	}

	public override void OnCreate()
	{
		base.OnCreate();
	}
}
```

Slika 5.10. Nasleđivanje klase Android.App.Application

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android
	package="Master.Master"
	android:versionCode="1"
	android:versionName="1.0"
	android:installLocation="auto">
	<uses-sdk android:minSdkVersion="21"
		android:targetSdkVersion="27" />
	<application android:allowBackup="true"
		android:label="@string/app_name"
		android:name="Master.App"
		android:theme="@style/AppTheme">
	</application>
</manifest>
```

Slika 5.11. AndroidManifest.xml aplikacije „Učigraonica“

5.3 Klase za rad sa bazom podataka

Rad sa bazom podataka ostvarije se preko tri klase uz pomoć SQLite baze:

- CDatabase – klasa koja inicijalizuje potrebne podatke, sadrži funkcije kojima se mogu vršiti upiti na samu bazu (CRUD)
- CImage – klasa koja prikazuje strukturu tabele u kojoj se nalaze informacije o pojmovima koji se pogađaju u igri čitanja i pisanja (slika 5.12)
- CRecord – klasa koja prikazuje strukturu tabele u kojoj se nalaze informacije o najboljim rezultatima postignutim u igrama matematike (slika 5.13)

```
public class CImage
{
    [PrimaryKey, AutoIncrement]
    public int Id { get; set; }
    public string Name { get; set; }
    public string SolutionLatin { get; set; }
    public string SolutionCyrilic { get; set; }
    public bool SolvedLatin { get; set; } = false;
    public bool SolvedCyrilic { get; set; } = false;
}
```

Slika 5.12. Struktura klase CImage

```
public class CRecord
{
    [PrimaryKey, AutoIncrement]
    public int Id { get; set; }
    public string GameId { get; set; }
    public int Result { get; set; }
    public int Time { get; set; }
}
```

Slika 5.13. Struktura klase CRecord

U okviru klase CDatabase se nalaze metode za upis, izmenu, čitanje i brisanje slogova iz baze podataka kojima se može pristupiti preko instance klase *App*. Primer poziva metode za upisivanje novog rezultata u bazu podataka pozivom metode *InsertRecord(CRecord record)* je dat na slici 5.14.

```
// Inicijalizacija i popunjavanje instance klase CRecord
CRecord record = new CRecord();
record.GameId = App.MathCode + (int)operacija;
record.Result = 10;
record.Time = vremeResavanja;
// App.db predstavlja instancu klase CDatabase
App.db.InsertRecord(record);
```

Slika 5.14. Primer rada sa bazom podataka preko klase App i njego polja CDatabase db

5.4 Postavljanje i čuvanje podešavanja

Struktura korisničkih podešavanja predstavljena je klasom *Preferences*. Instanca ove klase se nalazi u okviru klase *App* i dostupna je u celoj aplikaciji. Sva korisnička podešavanja se smeštaju u okviru *SharedPreferences* mehanizma pri svakoj izmeni samih podešavanja i pri ulasku u aplikaciju. *SharedPreferences* je interfejs koji pruža mogućnost čuvanja i čitanja podataka po principu ključ-podatak. Struktura klase *Preferences* je data na slici 5.15. dok su metode za upis i čitanje u *SharedPreferences* date na slici 5.16. Metoda *ReadSharedPreferences* se poziva pri svakom ulasku u aplikaciju radi popunjavanja instance klase *Preferences* dok se metoda *WriteSharedPreferences* poziva svaki put kada dođe do izmene u korisničkim podešavanjima.

```
public class Preferences
{
	public Preferences() { }
	public bool sound { get; set; }
	public bool vibration { get; set; }
	public int sound_volume { get; set; }
	public int sabiranje { get; set; } = 100;
	public int oduzimanje { get; set; } = 100;
	public int deljenje { get; set; } = 100;
	public int mnozenje { get; set; } = 100;
	public LangEnum language { get; set; } = LangEnum.Latinica;
}
```

Slika 5.15. Struktura klase Preferences za čuvanje korisničkih podešavanja

```
public void ReadSharedPreferences()
{
   ISharedPreferences prefs =
PreferenceManager.GetDefaultSharedPreferences(Current);

   preferences.sound = prefs.GetBoolean("sound", false);
   preferences.sound_volume = prefs.GetInt("sound_volume", 10);
   preferences.vibration = prefs.GetBoolean("vibration", false);
   preferences.sabiranje = prefs.GetInt("sabiranje", 100);
   preferences.oduzimanje = prefs.GetInt("oduzimanje", 100);
   preferences.mnozenje = prefs.GetInt("mnozenje", 100);
   preferences.deljenje = prefs.GetInt("deljenje", 100);
   var lang = prefs.GetInt("language", 0);
   preferences.language = (lang == 0) ? LangEnum.Cirilica : LangEnum.Latinica;
}

public void WriteSharedPreferences()
{
   ISharedPreferences prefs =
PreferenceManager.GetDefaultSharedPreferences(Current);

   var editor = prefs.Edit();

   editor.PutBoolean("sound", preferences.sound);
   editor.PutBoolean("vibration", preferences.vibration);
   editor.PutInt("sound_volume", preferences.sound_volume);
   editor.PutInt("sabiranje", preferences.sabiranje);
   editor.PutInt("oduzimanje", preferences.oduzimanje);
   editor.PutInt("mnozenje", preferences.mnozenje);
   editor.PutInt("deljenje", preferences.deljenje);
   if (preferences.language == LangEnum.Cirilica)
      editor.PutInt("language", 0);
   else
   editor.PutInt("language", 1);

   editor.Commit();
}
```

Slika 5.16. Metode za upis i čitanje podešavanje iz interfesa SharedPreferences

5.5 Aktivnosti

Za potrebe razvoja aplikacije “Učigraonica” definisane su sledeće Aktivnosti:

- *BaseActivity* – predstavlja baznu klasu za ostale Aktivnosti aplikacije, u sebi sadrži mogućnost postavljanja jezika aplikacije
- *MainActivity* – Aktivnost koja je vidljiva tokom celog toka aplikacije osim u slučaju prikazivanja Aktivnosti sa podešavanjima. Ova Aktivnost predstavlja početnu Aktivnost pri pokretanju aplikacije (*MainLauncher*)

- *SettingsActivity* – Aktivnost koja je vidljiva u sekciji sa podešavanjima

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
    <android.support.design.widget.TabLayout
        android:id="@+id/tabLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        app:tabIndicatorColor="#ffffff"
        app:tabSelectedTextColor="#ffffff"
        app:tabTextColor="#ffffff"
        android:layout_gravity="top">
        <android.support.design.widget.TabItem
            android:text="@string/igra"
            android:icon="@drawable/ic_play_w" />
        <android.support.design.widget.TabItem
            android:text="@string/zvuk"
            android:icon="@drawable/ic_sound_w" />
        <android.support.design.widget.TabItem
            android:text="@string/ostalo"
            android:icon="@drawable/ic_settings_w" />
    </android.support.design.widget.TabLayout>
    <android.support.v4.view.ViewPager
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

Slika 5.17. Datoteka šeme SettingsActivity.xml u kojoj se koristi ViewPager sa TabLayout-om

U okviru *SettingsActivity* korišćen je pogled *ViewPager* sa tri različite sekcije: podešavanje težine matematičkih operacija, podešavanja zvuka i brisanje baze ostvarenih rekorda. Svaka od sekcija pogleda *ViewPager* je instanca klase Fragment, odnosno *PlaySettingsFragment*, *SoundSettingsFragment* i *OtherSettingsFragment*. U samom pogledu *ViewPager* korišćen i je pogled *TabLayout* za prikaz tabova (jezičaka) zajedno sa ikonicama i tekstom. Svaki *ViewPager* pogled ima i svoj prateći adapter (*SettingsPagerAdapter*) koji brine o instanciranju potrebnih Fragmenata i njihovom prikazu pri kretanju kroz *ViewPager*. XML datoteka šeme u kojoj je *ViewPager* zajedno sa *TabLayout*-om data je na slici 5.17. dok je primer adaptera prikazan na slici 5.18.

```
public class SettingsPagerAdapter : FragmentPagerAdapter
{
  Activity activity;
  Android.Support.V4.App.FragmentManager fragmentManager;
  Android.Support.V4.App.Fragment fragmentPlay, fragmentSound, fragmentOther;

  public SettingsPagerAdapter(Activity activity,
Android.Support.V4.App.FragmentManager fm) : base(fm)
  {
      this.activity = activity;
      this.fragmentManager = fm;
  }

  public override int Count { get { return 3; } }

  public override int GetItemPosition(Java.Lang.Object objectValue)
  { return base.GetItemPosition(objectValue); }

  public override Android.Support.V4.App.Fragment GetItem(int position)
  {
      if (position == 0)
      {
        fragmentPlay = new PlaySettingsFragment();
        return fragmentPlay;
      }
      else if (position == 1)
      {
        fragmentSound = new SoundSettingsFragment();
        return fragmentSound;
      }
      else
      {
        fragmentOther = new OtherSettingsFragment();
        return fragmentOther;
      }
  }
}
```

Slika 5.18. SettingsPagerAdapter – adapter za rad sa ViewPager-om

5.6 Promena jezika aplikacije

Aplikacija “Učigraonica” omogućava upotrebu dva različita pisma, latinice ili ćirilice. Samim tim potrebno je da prikaz svog teksta bude usaglašen sa izborom pisma. To se postiže upotrebom dve datoteke *String.xml* od kojih se jedna, sa niskama pisanim latinicom, nalazi u direktorijumu *Resources/values*, dok se druga, sa niskama pisanim ćirilicom, nalazi u direktorijumu *Resources/values-sr*. Ovim vrlo jednostavnim mehanizmom, odnosno upisom svih potrebnih niski u dve odvojene datoteke u odgovarajućem pismu postiže se lak prelaz aplikacije iz jednog pisma u drugo pismo.

Postavljanjem odgovarajućih parametara u okviru aplikacije operativni sistem automatski određuje koja od navedene dve datoteke će biti u upotrebi. Primer podešavanja aplikacije da radi sa jednim ili drugim pismom je dat na slici 5.19.

```
// upis korisničkih podešavanja u SharedPreferences
App.preferences.language = LangEnum.Cirilica;
App.Current.WriteSharedPreferences();

// izbor pisma - ćirilica
Java.Util.Locale.Default = new Locale("sr", "RS");
Resources.Configuration.Locale = Java.Util.Locale.Default;
Resources.UpdateConfiguration(Resources.Configuration,
Resources.DisplayMetrics);
// nakon navedenih poziva poziv metode GetString(Resource.String.igraj)
// vratiće vrednost “ИГРАЈ“

App.preferences.language = LangEnum.Latinica;
App.Current.WriteSharedPreferences();

// izbor pisma - latinica
Java.Util.Locale.Default = App.default_locale;
Resources.Configuration.Locale = Java.Util.Locale.Default;
Resources.UpdateConfiguration(Resources.Configuration,
Resources.DisplayMetrics);
// nakon navedenih poziva poziv metode GetString(Resource.String.igraj)
// vratiće  vrednost “IGRAJ“
```

Slika 5.19. Promena jezika aplikacije

5.7 *SharedTransitions* u okviru Fragmenta

U teorijskom delu rada predstavljen je mehanizam animacija zajedničkih elemenata pri promeni ekrana aplikacije. Taj princip se zasnivao na prelasku iz jedne Aktivnosti u drugu. Česta je pojava da se sistem animacija zajedničkih elemenata primeni i u slučaju kada dva Fragmenta sadrže zajedničke elemente.

U aplikaciji “Učigraonica” *ReadingAndWritingFragment* i *GuessingImageFragment* imaju zajednički element sliku pojma koji treba pogoditi.

ReadingAndWritingFragment – u nastavku RAWF, prikazuje listu svih pojmova sa mogućnošću izbora pojma koji igrač želi da pogađa.

GuessingImageFragment - u nastavku GIF, prikazuje ekran za pogađanje pojma zajedno sa slikom samog pojma.

Zajednički elementi, u ovom slučaju dva pogleda tipa *ImageView*, moraju imati identičnu vrednost polja *TransitionName*. S obzirom da se u RAWF radi o listi pojmova, svaki red, odnosno svaki pogled ImageView mora imati jedinstven *TransitionName*. Da bi se obezbedilo da to bude slučaj kao vrednost polja *TransitionName* uzima se naziv pojma iz baze podataka pojmova. Nakon odabira pojma iz liste (dodirom na željeni red) u RAWF potrebno je taj Fragment zameniti instancom Fragmenta GIF i prikazati animacije zajedničkih elemenata. Primer jednog takvog poziva dat je na slici 5.20.

```
// zajednički element u početnom Fragmentu
ImageView Slika = clickedRow.Slika;

// vrednost polja TransitionName
string transitionName = Slika.TransitionName;

GuessingImageFragment gif = new
         GuessingImageFragment(mAdapter.clickedImage.Id);

Activity.FragmentManager.BeginTransaction()
        .Replace(App.fragmentContainer.Id, gif, "reading_and_writing")
// dodavanje zajedničkog elementa i vrednosti polja TransitionName
        .AddSharedElement(Slika, transitionName)
        .AddToBackStack(null)
        .Commit();
```

Slika 5.20. Primer zamene Fragmenta novim Fragmentom uz korišćenje zajedničkih elemenata

U novom Fragmentu, u ovom slučaju GIF, dovoljno je postaviti vrednost polja *TransitionName* nad instancom zajedničkog elementa tipa *ImageView*. Kao i u slučaju zajedničkih elemenata kod Aktivnosti, moguća je upotreba izabranih animacija.

5.8 RecyclerView za prikaz liste pojmova

O *RecyclerView* elementu je već bilo reči u teorijskom delu ovog rada. On predstavlja optimizovanu i unapređenu verziju elementa *ListView* jer već implementira obrazac *ViewHolder*. Lista pojmova u aplikaciji "Učigraonica" je prikazana upravo koristeći *RecyclerView* i implementacijom odgovarajućeg adaptera. Da bi se koristio *RecyclerView* potrebno je definisati sledeće objekte:

- *RecyclerView.Adapter* – klasa koja služi za popunjavanje redova liste podataka
- *RecyclerView.LayoutManager* – klasa koja služi za prikaz samih redova, njihovih pozicija, da li su elementi prikazani kao mreža ili jedan ispod drugog itd.

Slika 5.21. prikazuje upotrebu navedenih klasa pri inicijalizaiciji objekta tipa *RecyclerView*.

```
RecyclerView mRecyclerView =
rootView.FindViewById<RecyclerView>(Resource.Id.recyclerView);

var mLayoutManager = new LinearLayoutManager(Activity);
mRecyclerView.SetLayoutManager(mLayoutManager);

// lista svih pojmova
var listOfItems = App.db.GetAllImages();

var mAdapter = new RecyclerViewAdapter(listOfItems, Activity);
mRecyclerView.SetAdapter(mAdapter);
```

Slika 5.21. Inicijalizacija objekta tipa RecycelerView

Klasa adapter koja nasleđuje *RecyclerView.Adapter* klasu mora imati metode:

- *OnCreateViewHolder* – u okviru koje se vrši inicijalizacija objekta tipa *ViewHolder* i dodeljuje mu se odgovarajuća resursna datoteka šeme
- *OnBindViewHolder* – u okviru koje se vrši povezivanje podataka sa elemetima tog reda

Klasa *RecyclerViewAdapter* je prikazana na slici 5.22. dok je *ViewHolder* korišćen u razvoju aplikacije prikazan na slici 5.23.

```
public class RecyclerViewAdapter : RecyclerView.Adapter
{
    public event EventHandler<int> ItemClick;
    List<CImage> items;
    Activity activity;
    public CImage clickedImage;
    public ImageView sharedImageVIew;
    public int ClickedPosition;

    public RecyclerViewAdapter(List<CImage> pItems, Activity pActivity)
    {
        items = pItems;
        activity = pActivity;
    }

    public override RecyclerView.ViewHolder OnCreateViewHolder(ViewGroup
parent, int viewType)
    {
        CImage item = null;
        string layout = "";
        if (items != null)
            item = items[0];

        View view = null;
        if (item != null)
        {
          view = activity.LayoutInflater
                 .Inflate(Resource.Layout.image_view_holder, null);
          LinearLayout.LayoutParams LLParams = new LinearLayout
                 .LayoutParams(LinearLayout.LayoutParams.MatchParent,
                  LinearLayout.LayoutParams.WrapContent);
          view.LayoutParameters = LLParams;
        }

        var vh = new RecyclerViewHolder(view, activity, OnClick);
        return vh;
     }

     public override void OnBindViewHolder(RecyclerView.ViewHolder holder, int
position)
     {
       var item = items[position];
       RecyclerViewHolder vh = holder as RecyclerViewHolder;

       vh.Naslov.Text = "";
       if (App.preferences.language == LangEnum.Latinica)
       {
         if (item.SolvedLatin)
         {
           vh.Naslov.Text = item.SolutionLatin.ToUpper();
           vh.grayLayer.Visibility = ViewStates.Gone;
         }
         else
         {
```

```
      for (int i = 0; i < item.SolutionLatin.Length; i++)
          vh.Naslov.Text += "__ ";

      vh.grayLayer.Visibility = ViewStates.Visible;
    }
  }
  else
  {
    if (item.SolvedCyrilic)
    {
      vh.Naslov.Text = item.SolutionCyrilic.ToUpper();
      vh.grayLayer.Visibility = ViewStates.Gone;
    }
    else
    {
      for (int i = 0; i < item.SolutionCyrilic.Length; i++)
      {
        vh.Naslov.Text += "__ ";
        vh.grayLayer.Visibility = ViewStates.Visible;
      }
    }
  }

  vh.Slika.SetImageResource(activity.Resources.GetIdentifier(item.Name,
"drawable", activity.PackageName));
  vh.Slika.TransitionName = item.Name;

  if (item.Id < 70)
  {
     vh.Star1.SetImageResource(Resource.Drawable.ic_star_g);
     vh.Star2.SetImageResource(Resource.Drawable.ic_star);
     vh.Star3.SetImageResource(Resource.Drawable.ic_star);
  }
  else if (item.Id < 150)
  {
     vh.Star1.SetImageResource(Resource.Drawable.ic_star_g);
     vh.Star2.SetImageResource(Resource.Drawable.ic_star_g);
     vh.Star3.SetImageResource(Resource.Drawable.ic_star);
  }
  else
  {
     vh.Star1.SetImageResource(Resource.Drawable.ic_star_g);
     vh.Star2.SetImageResource(Resource.Drawable.ic_star_g);
     vh.Star3.SetImageResource(Resource.Drawable.ic_star_g);
  }
}

public override int ItemCount
{
  get
  {
     if (items != null)
        return items.Count;
     else
        return 0;
  }
```

```
  public void OnClick(int position)
  {
    clickedImage = items[position];
    ClickedPosition = position;
    if (ItemClick != null)
        ItemClick(this, position);
  }
}
```

Slika 5.22. Implementacija RecyclerViewAdapter klase

```
public class RecyclerViewHolder : RecyclerView.ViewHolder
{
    public LinearLayout grayLayer { get; private set; }
    public TextView Naslov { get; private set; }
    public ImageView Slika { get; private set; }
    public ImageView Star1 { get; private set; }
    public ImageView Star2 { get; private set; }
    public ImageView Star3 { get; private set; }

    public RecyclerViewHolder(View itemView, Activity activity, Action<int>
listener) : base(itemView)
    {
       grayLayer = ItemView.FindViewById<LinearLayout>(Resource.Id.grayLayer);
       Naslov = ItemView.FindViewById<TextView>(Resource.Id.nazivSlike);
       Slika = ItemView.FindViewById<ImageView>(Resource.Id.slika);
       Star1 = ItemView.FindViewById<ImageView>(Resource.Id.star1);
       Star2 = ItemView.FindViewById<ImageView>(Resource.Id.star2);
       Star3 = ItemView.FindViewById<ImageView>(Resource.Id.star3);

       itemView.Click += delegate
       {
          listener(base.Position);
       };
    }
}
```

Slika 5.23. ViewHolder korišćen za prikaz jednog pojma u listi pojmova

5.9 Upotrebljene biblioteke

Pri razvoju ove aplikacije korišćene su biblioteke FButton i TransitionsEverywhere. FButton omogućava korišćenje posebno dizajniranih dugmadi čiji se izgled može videti na slci 5.24. dok TransitionsEverywhere olakšava postavljanja i upotrebu animacija. Na slici 5.25. se može videti par primera upotrebe TransitionsEverywhere u različitim situacijama (animacije prilikom promene teksta, boje, pozicije itd.).

Slika 5.24. Prikaz dugmeta iz biblioteke FButton u početnom stanju i kada je pritisnuto

```
// animacija u kojoj pri promeni teksta stari tekst “izbledi” a zatim se novi
// postepeno pojavi
// rootView – pogled u kom se nalazi pogled koji menja tekst
TransitionManager.BeginDelayedTransition((ViewGroup)rootView, new
            ChangeText().SetChangeBehavior(ChangeText.ChangeBehaviorOutIn));


// animacija u kojoj pogled koji želimo da sakrijemo izbledi, ili ukoliko
želimo
// da ga prikažemo se postepeno pojavi
// rootView – koreni pogled u kom se odigrava promena
// textView – pogled koji želimo da sakrijemo/prikažemo
boolean visible = true;
TransitionManager.BeginDelayedTransition(rootView);
textView.Visibility = visible ? ViewStates.Visible : ViewStates.Gone;
```

Slika 5.25. Upotreba biblioteke TransitionsEverywhere

6 Zaključak

Tržište “pametnih” mobilnih telefona i tableta je u konstantom porastu dok prisustvo konkurencije dovodi do usavršavanja i poboljšanja performansi navedenih uređaja. Virtuelna realnost, veštačka inteligencija, grafički zahtevne video igre su samo neka od polja gde su mobilni uređaji našli primenu. Broj korisnika ali i raznovrsnost uređaja uslovljava izmene i u samom operativnom sistemu Android, procesu razvoja aplikacija, ujedno nastaju novi principi, radni okviri, obrasci. Programerima su na raspolaganju nove metode i mogućnosti za implementaciju u aplikacijama ali se i povećava potrebno znanje da bi razvijana aplikacija pratila trendove napretka tehnologije.

Iako je Android operativni sistem prošao kroz mnoga unapređenja i nove verzije, i dalje se razvija od strane kompanije *Google*. Takođe, tržišni udeo ovog operativnog sistema je u stalnom usponu što povećava potražnju za programerima koji poznaju navedene tehnologije.

Korisnički interfejs, sistem sa kojim korisnici interaguju sa aplikacijom, predstavlja veoma bitan aspekt pri razvoju. Dobro definisan i funkcionalan korisnički interfejs doprinosi boljoj prihvaćenosti i zadovoljstva među korisnicima aplikacije. Glavni doprinos ovog rada je bliže upoznavanje sa principima, pravilima, konceptima ali i tehnologijama koje su potrebne za definisanje jednog takvog korisničkog interfejsa. Sve ovo je ostvareno kroz upoznavanje sa samim operativnim sistemom, njegovom arhitekturom, osnovnim principima programiranja korisničkog interfejsa, pregledom najnovijih tehnologija, ali i kroz savete i preporuke date u radu.

Cilj implementacije aplikacije „Učigraonica“ je prikazivanje velikog broja principa, koncepta i radnih okvira za programiranje i dizajniranje korisničkog interfejsa na platformi Android o kojima je bilo reči u prethodnim poglavljima ovog rada. Motivacija za razvoj aplikacije je mogućnost da deca uzrasta od 6 do 12 godina kroz igru, na lak i zabavan način, savladaju čitanje i pisanje latinice i ćirilice, kao i osnovne matematičke operacije. Aplikacija je razvijana po principu otvorenog koda i izvorni kod aplikacije je moguće pronaći na adresi: `https://github.com/panucci/Ucigraonica-Master`

7 Literatura

[1] Eric Hellman, "*Android Progamming: Pushing the Limits*", 2014.

[2] Mark L. Murphy, "*The Busy Coder's Guide to Android Development*", 2011.

[3] Ronan Schwarz, Phil Dutson, James Steele, Nelson To, "*The Android Developer's Cookbook: Building Applications with the Android SDK , Second Edition*", 2013.

[4] Dave Smith, Jeff Friesen, "*Android Recepies: A Problem-Solution Approach, Second Edition*", 2011.

[5] Counterclockwise: Increasing the resolution:
https://www.gsmarena.com/counterclockwise_increasing_the_resolution-news-26065.php
Poslednji put pristupljeno 10.9.2018.

[6] Android (operating system):
https://en.wikipedia.org/wiki/Android_(operating_system)
Poslednji put pristupljeno 10.9.2018.

[7] Global mobile OS market share:
https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/
Poslednji put pristupljeno 10.9.2018.

[8] Distribution dashboard
https://developer.android.com/about/dashboards/
Poslednji put pristupljeno 10.9.2018.

[9] Platform Architecture:
https://developer.android.com/guide/platform/
Poslednji put pristupljeno 10.9.2018.

[10] Create an Android library:
https://developer.android.com/studio/projects/android-library
Poslednji put pristupljeno 10.9.2018.

[11] Material Design:
https://material.io/
Poslednji put pristupljeno 10.9.2018.